Da Vinci

Definition & Design of Information Systems

Version of: 17-07-2004

Contents

Preface **7**

1 Introduction **9**

1.1 The information systems area 9

1.1.1 Information systems 9

1.1.2 Information system development 10

1.2 Challenges for information system development 11

1.2.1 Evolution is a constant 11

1.2.2 Complexity; the gravitational force of software construction 13

1.3 Architecture-driven information systems development 14

1.3.1 Architecture 14

1.3.2 Alignment 16

1.3.3 Architecture-driven information system development 17

1.4 Structuring the domain 17

1.4.1 Methodological framework 17

1.4.2 Information systems are systems too 19

1.4.3 Structure of this text-book 20

2 Systems and their Architecture **25**

2.1 System concepts 25

2.1.1 Introduction 25

2.1.2 Core concepts 26

2.1.3 Systems 27

2.1.4 Sub-systems 30

2.1.5 System views 31

2.1.6 System evolution 32

2.2 System development 33

2.2.1 Key processes 33

2.2.2 Stakeholders and their concerns 35

2.3 System viewpoints 36
2.3.1 The need for viewpoints 37
2.3.2 Viewpoint dimensions 38
2.4 System architecture 44
2.4.1 The need for architecture 45
2.4.2 A pragmatic definition of architecture 46
2.4.3 Architecture-driven system development 47

3 Systems Modelling **51**
3.1 Refinements between system views 51
3.2 Abstraction mechanisms 52
3.2.1 General properties of abstractions 52
3.2.2 Typing 53
3.2.3 Generalisation 53
3.2.4 Encapsulation 53
3.2.5 Implementation 53
3.3 Towards a generic viewpoint 55

4 System Definition **61**
4.1 Understanding system needs 61
4.1.1 Requirements 61
4.1.2 System quality 61
4.1.3 A pragmatic context for requirements 63
4.1.4 Alignment 65
4.2 Describing requirements 65
4.2.1 Requirement types 65
4.2.2 Appraisal of requirements 66
4.2.3 Good requirements 67
4.2.4 Requirements description languages 68
4.3 Eliciting requirements 68
4.3.1 Activities 68
4.3.2 Challenges 68

5 System Design **71**

5.1 Design description languages 71

5.2 System design as a refinement process 72

5.2.1 Refinement 72

5.2.2 Dimensions of refinement 73

5.2.3 Refinement levels 73

5.2.4 Design decissions 73

5.3 Design principles 74

5.4 Responsibility based design 77

Complete Bibliography **81**

Glossary of terms **85**

Glossary of symbols **93**

Subject Index **95**

A Mathematical Notations **99**

A.1 Sets 99

A.2 Functions 99

A.3 Relations 99

B GNU Free Documentation License **101**

B.1 Applicability and Definitions 101

B.2 Verbatim Copying 102

B.3 Copying in Quantity 102

B.4 Modifications 103

B.5 Combining Documents 104

B.6 Collections of Documents 104

B.7 Aggregation With Independent Works 105

B.8 Translation 105

B.9 Termination 105

B.10 Future Revisions of This License 105

Preface

Version:
17-07-2002

The intention of this document is that it evolves into a text-book on architecture-driven information systems development for the subject "Information Systems Architecture" at the University of Nijmegen. The current version does require the reader to have a considerable willingness to read between the lines, as the text is still rather stenographic in nature and while parts of the text-book are still sketchy. Over the next few years, the text should be improved. The intended 'life-cycle' of this document over the next few years is:

2002-2003 Provide initial content, mainly based on pre-existing contents, during the first run of the subject Architecture & Alignment.

2003-2004 Bring content more up-to-date with state-of-the-art.

2004 & beyond Stepwise addition of more fundamental theory.

My personal interest in the field of architecture-driven development of information systems was raised in 1997/1998 when I was employed as a consultant by Origin (now called Atos-Origin) and was first exposed to the field. This lead to a first document referred to as "Da Vinci" [Pro98]. Management at Origin gave me the freedom – for which I am still grateful – to work on "Da Vinci", drawing together many valuable ideas on architecture-driven development of information systems from several colleagues from within Origin. The resulting document stipulated a shared Origin vision on architecture-driven development of information systems. The name "Da Vinci" is not some artificial acronym. The name was chosen by me to honour an inspiring artist, scientist, inventor and architect.

To acknowledge the origins of my interest in the field, I have chosen to also use the name "Da Vinci" on this new document.

During the development of this text-book and the associated lectures, I intend to consult people from industry who work on in the field of information systems architectures in practice. The current list of practitioners, who actively provide feedback, consists of:

1. Araminte Bleeker, Luminus
2. Hans Bosma, Ordina
3. Jan Campschroer, Ordina
4. Steven in't Veld, AIM
5. Denis Verhoef, Ordina

Please sent any comments to the author at: `E.Proper@acm.org`

Chapter 1

Introduction

Version:
17-07-2004

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.

There is another theory which states that this has already happened.

From: "The Restaurant at the End of the Universe", Douglas Adams, Pan Books Ltd.

1.1 The information systems area

May different terminologies are used in discussing different aspects from the information systems area. In this introduction, we start by first defining a basic terminology which we will use, and refine, throughout this text-book. In doing so, we will draw from three main sources. We will (mainly) draw terminology pertaining to:

- architectures from the *IEEE recommended practice for software intensive systems* [IEE00],
- systems and information systems from the *framework of information system concepts* [FVSV+98],
- development processes from the *information services procurement library* [FV99, Pro01].

1.1.1 Information systems

As stated in [FVSV+98], "*information systems*" concerns the use of "*information*" by individual or groups of people in organisations, in particular through computer-based systems. In line with [FVSV+98], we use the term "*organisation*" here, and throughout this textbook, in the most general sense. Not only large companies are meant. One-man companies, profit- and non-profit-oriented organisations, clusters of companies interacting with each other, even the community of all Internet users and similar communities, may all be considered organisations.

The concept of information system can roughly be defined as that aspect of an organisation that provides, uses and distributes information. An information system *may* contain computerised sub-systems to automate certain elements. Some information system may not even be computerised at all. A filing cabinet used to store and retrieve several dossiers is, in essence, an information system. The kind of information systems we are interested in, however, are indeed presumed to have some computerised core parts.

What we may perceive to be an information system, may vary highly in terms of their scope. Some examples would be:

- Personal information appliances, such as electronic agenda's, telephone registries in mobile phones, etc.
- Specific information processing applications.
- Enterprise wide information processing.
- Value-chain wide information processing.

In practice, the concept of "*information system*" is used quite differently by different groups of people. It seems (see e.g. [FVSV+98]) to be interpreted in at least three different ways:

- As a technical system, implemented with computer and telecommunications technology.
- As a social system, such as an organisation, in connection with its information processing needs.
- As a conceptual system (i.e. an abstraction of either of the above).

A more precise definition (based on [FVSV+98]) of the way we view the concept of information system system is:

> **Information system –** a sub-system of an organisational system, comprising the conception of how the communication and information-oriented aspects of an organisation are composed and how these operate, thus leading to a description of the (explicit and/or implicit) communication-oriented and information-providing actions and arrangements existing within the organisational system.

This definition refers in its term refers to the concept of organisational system, which is essentially a systemic view on an organisation:

> **Organisational system –** a special kind of system, being normally active and open, and comprising the conception of how an organisation is composed and how it operates (i.e. performing specific actions in pursuit of organisational goals, guided by organisational rules and informed by internal and external communication), where its systemic properties are that it responds to (certain kinds of) changes caused by the system environment and, itself, causes (certain kinds of) changes in the system environment.

Using the definition of information system, we may specialise this to its computerised parts as follows:

> **Computerised information system –** a sub-system of an information system, whereby all activities within that sub-system are performed by one or several computer(s).

In section 2.1, we will be able to provide an even more precise definition.

1.1.2 Information system development

Most information systems, in particular computerised ones, do not appear out of the blue. They need to be developed using some development process. We view the development of an information system as involving four processes:

Definition process – a process aiming to identify all requirements that should be met by the system and the system description.

In literature this process may also be referred to as requirements engineering.

Design process – a process aiming to design a system conform stated requirements. The resulting system design may range from high-level designs, such as an strategy or an architecture, to the detailed level of programming statements or specific worker tasks.

Construction process – a process aiming to realise and test a system that is regarded as a (possibly artificial) artifact that is not yet in operation.

Installation process – a process aiming to make a system operational, i.e. to implement the use of the system by its prospective users.

In these definitions, the term system means to refer to either an organisational system, information system or computerised information system. The definitions of the construction process and the installation process are conform the definitions used in [FV99]. Conform [FV99], the definition process and design process are collectively referred to as the description process. However, in the context of this text-book, a clear distinction between the definition of the future system in terms of its requirements, and the actual system design is indeed needed.

The concept of information system development may thus be defined as:

Information system development – a process involving the execution of four sub-processes: definition, design, construction and installation of an information system. Processes that may be executed sequentially, interleaved, or in parallel.

Note that, these processes in no way need to be executed linearly. Most practical situations require a non-linear execution of these processes, e.g. an evolutionary or incremental approach.

In this text-book we are mainly concerned with the definition and design processes, as these are influenced the most by the use of architecture.

1.2 Challenges for information system development

1.2.1 Evolution is a constant

The prevailing conditions under which most organisations currently operate have a tendency to evolve constantly. Reduced protectionism, de-monopolisation of markets, deregulation of international trade, privatisation of state owned companies, increased global competition, cross-border merges, the emergence of new trade blocks, the introduction of common currencies, all contribute toward this increasingly dynamic business environment. Developments that are fuelled even more by the advances of eCommerce, Networked Business, Virtual Enterprises, etc. To improve their chances for survival, organisations need the ability to quickly adapt themselves to such socio-economic developments.

Organisations make use of (largely computerised) information systems to provide in their information processing needs. When an organisation evolves, these information systems should be able to co-evolve in a natural way [Pro94]. In practice, this has proven to be a difficult task, in particular where it concerns the computerised (parts of the) information systems, i.e. information technology (information technology). Studies, such as the ones reported in [LS80, BP88, NP90] have shown that a large part of the costs associated to computerised information systems are spent on such modifications.

Ideally, information technology should empower an organisation with the ability to go out and seek new challenges. However, one of the current dilemmas of information technology seems to be that in most cases it smothers an organisation's ability to change rather than supporting it. While it is quite reasonable to say that advanced computerised information systems should lead to revolutionary improvements

in the flexibility and effectiveness of organisations, organisations still find themselves anchored to their pre-existing information systems. Quite often, these systems are the embodiment of the prevailing cultures and structures of the organisation's past. These systems tend to have an almost tangible monolithic nature that would be a feast to software archologists.

Example 1.2.1

Two concrete examples of such application domains with rapidly changing information needs are:

Taxes In most nations the (income) tax laws change quite often as they are used both to manage the economy, and to finance government policy.

Software firms developing and maintaining software for the calculation of, say, income taxes for company employees, have to change their software each time the government changes income tax. The change in information needs is caused by the changed laws.

Recent examples of changes in tax laws can be found in many of the nations of the European Community, due to the tax harmonisations brought about by the unification process.

Insurance Most insurance companies change the rules by which policy prices are calculated regularly. These changes are usually intended to improve the competitiveness of the policy pricing, for instance, the no-claims bonus for not claiming damages in the case of car insurances. As a result, the software for calculating the policy prices has to be changed to cope with any extra information required (history of damages claimed), as well as new formulas to perform the calculation. In this case, the changes in the information need are caused by marketing arguments.

An organisation can deal with changes in their environment in a variety of ways. While some may try and continue their business *as usual*, others may choose to embrace the new developments and try to exploit their potential to their fullest. Neither approach is a guaranteed way to success or failure. Embracing new developments too early may lead to organisational chaos and decline, while waiting too long may result in missed business opportunities. The role of information technology can be characterised as a position between two undesirable extremes. On the one extreme, information technology can completely restrain organisational change. In the other extreme, it can (try to) bring about organisational changes in a pace which is far too high. The latter situation can be compared to putting a Ferrari engine in a VW Beetle, while the driver is used to easing along at maximum speeds of 80 to 100 km/h. In the first extreme, information technology appears to smother any organisational change. This seems to be one of the dilemmas of information technology. In the second extreme, information technology will drive an organisation to a pace of change that goes beyond the speed its organisational fabrics can manage. Some organisations are just not ready to cope with the profound changes brought about by a technology push. For example, organisations that have only just reached a stage at which they are confident with the use of some basic information technology to automate the processing of orders might simply not survive a quick move towards electronic commerce.

Already in [Kee91] and [TC93], an elaborate discussion can be found on the changes in context and culture that are occurring inside organisations as well as in their environments as a result of different socio-economic changes in combination with technological developments in information technology. Tapscott [TC93] proposes an architectural approach as a solution to make the needed changes to the organisational structure and in particular information technology. These new demands on information technology in the new and rapidly evolving world, can be summed up by quoting [Tap96]:

> *In the past an architecture was really the design of a system that had been created to meet specific application needs. In the new business environment, organisations have little idea what their application needs will be in two, let alone five or ten years. Consequently, we need architectures that can enable the exploitation of unforeseen opportunities and meet unpredictable needs.*

Development of information systems in such rapidly evolving contexts becomes like shooting at a moving target [PW95a, PW94, PW95b]. This requires us to look at organisations and their information systems as evolving systems [Pro94] that are in a constant state of co-evolution.

Ideally, business strategists should be able to focus solely on the development of a business, while information technology plays the role of a catalyst. Tapscott [TC93] argues that organisations move between different levels of organisational development, improving their ability to cope with changes/evolution in their environment. Information technology should act as one of the essential enablers of this process. In figure 1.1, taken from [TC93], these levels have been depicted. By redesigning their business processes, organisations will be able move to a situation in which teams can perform better. This development leads to high-performance business teams, where the focus is on the use of Information technology to enable teams to perform business functions. This requires a shift away from a hierarchical view on organisations to a more team based view. The next shift involves the integration of the business teams to an integrated whole, leading to an integrated organisation. The role of information technology in these cases, is increasingly one of being an enabler; from cost centre to profit centre. By linking their systems to other organisations, in particular customers and suppliers, an organisation can finalise the paradigm shift, and become an extended enterprise.

Figure 1.1: The enabling effect of information technology

All these shifts and developments pose new requirements on the information technology function. These shifts, however, will not come from information technology alone. As Michael Hammer argued in his seminal paper: *Re-engineering work: don't automate, obliterate* [Ham90], improving the flexibility and efficiency of business processes is more than just using information technology to make it go faster. Information technology is only part of the answer, it should not be looked upon as the sole bringer of solutions, but rather as an enabler. Investments in a re-engineering of the business and better business-IT alignment will be needed just as well. However, organisational change should be business-driven, and not (solely) information technology-driven. The outcry for information technology that enables organisational change rather than inhibits it is clearly louder than ever. Information technology, and associated application development, should therefore be driven by the needs of the business.

1.2.2 Complexity; the gravitational force of software construction

In architectural design of man-made constructions, a pivotal role is played by the struggle with *gravity*. In the software architectures for computerised information systems, this role seems to be played by the struggle with *complexity*. We can all see around us how the software systems we develop, start to break

down under their own weight. The complexity of these systems has already reached a point where no single person is able to grasp all details of a systems working. The increase of complexity is fuelled by our thirst for ever more functionality. We want our computerised information systems to do more and more work for us. We require them to increase their scopes from a single unified organisation to cover entire coalitions of networked organisations, leading all sorts of interoperability problems. The constant pressure to evolve, as discussed above, makes this situation even more serious, leading to unreliable software, high maintenance costs, maintenance backlogs, etc.

The seriousness of software complexity has been reported in several sources. For example, in [Coc01], the following can be read:

> *What's the most important problem in computer science?*
>
> *Languages, tools, programmers?*
>
> *Well, according to a growing number of researchers and computer users, it's software complexity.* "We've known about this problem for 40 years," *says Alfred Spector, vice-president at IBM Research.*
>
> "This is probably the number one problem...It can't go on."

Brooks, in his Mythical Man-Month review [Bro95], mentions three distinctive concerns of software engineering, of which the last one reads: *How to maintain intellectual control over* **complexity** *in large doses.* Brooks, furthermore, writes:

> *The tar pit of software engineering will continue to be sticky for a long time to come. One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate of man's handiworks.*

1.3 Architecture-driven information systems development

1.3.1 Architecture

During the last decade, software architecture has received an increasing amount of attention in the software engineering community; both from research and from industry. The rationale behind this interest is that software architecture provides a number of important benefits [BCK98]:

- It is a vehicle for communication among stakeholders. A software architecture, often depicted graphically, can be communicated with end users, the client, designers, and so on.

  By developing scenarios of anticipated use, relevant quality aspects can be analysed and trade-offs can be discussed with various stakeholders.

- It captures early design decisions, both functional aspects as well as quality aspects. In a software architecture, the global structure of the system has been decided upon, through the explicit assignment of functionality to components of the architecture.

  These early design decisions are important since their ramifications are felt in all subsequent phases. It is therefore paramount to assess their quality at the earliest possible moment. By evaluating the architecture, a first and global insight into important quality aspects can be obtained. The global structure decided upon at this stage also structures development: the work-breakdown structure may be based on the decomposition chosen at this stage, testing may be organized around this same decomposition, and so on.

These advantages are not only limited to software (as it may be found in computerised information systems), but equally well relate to most types of systems.

Several definitions of the concept of architecture, in the context of systems, exist. According to [Mer03], architecture is:

1 the art or science of building; *specifically*: the art or practice of designing and building structures and especially habitable ones,

2a formation or construction as or as if as the result of conscious act,

2b a unifying or coherent form or structure,

3 architectural product or work,

4 a method or style of building,

5 the manner in which the components of a computer or computer system are organized and integrated.

In the context of software-intensive systems, an IEEE working group [IEE00] has provided a definition of architecture. The resulting definition is, indeed, in line with the general definition of architecture (in particular interpretations 1 and 5), but specialises it to software-intensive systems. As (computerised information systems are generally software-intensive systems, we shall use this definition throughout this text-book:

> **Architecture –** the fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its evolution and design.

As an architecture provides guidelines for the (detailed) design and evolution of a system, it is a powerful *means* to approach evolution and complexity of organisations, their context and their information systems. Note that *approach* not necessarily mean *control* of every detail.

Architectures are usually expressed in terms of architectural descriptions, essentially design descriptions pertaining to a systems architecture. In [IEE00] the following potential uses of architectural descriptions are identified:

- Expression of the system and its (potential) evolution.
- Analysis of alternative architectures.
- Business planning for transition from a legacy architecture to a new architecture.
- Communications among organizations involved in the development, production, fielding, operation, and maintenance of a system.
- Communications between acquirers and developers as a part of contract negotiations
- Criteria for certifying conformance of implementations to the architecture.
- Development and maintenance documentation, including material for reuse repositories and training material.
- Input to subsequent system design and development activities.
- Input to system generation and analysis tools.
- Operational and infrastructure support; configuration management and repair; redesign and maintenance of systems, subsystems, and components.
- Planning and budget support.

- Preparation of acquisition documents (e.g., requests for proposal and statements of work).
- Review, analysis, and evaluation of the system across the life cycle.
- Specification for a group of systems sharing a common set of features, (e.g., product lines).

1.3.2 Alignment

In information systems development, another pivotal role is played by alignment. In literature this is usually referred to as business-IT alignment or as strategic alignment [TC93, Kee91, PB89, HV93].

The importance of a good alignment between business and information technology is discussed by several authors. For example, Tapscott & Caston [TC93], [Boa99] and Keen [Kee91], and they are certainly not the first in doing this, provide extensive motivations. In [PB89], Parker and Benson already discussed the need for strategic alignment between business and information technology strategies. They argued that information technology planning and strategic considerations are part of a circular process as depicted in Figure 1.2. In this process, a distinction is made between the business domain on the one side, and the technology domain on the other side. Business planning drives how an enterprise will be organised, which should on its turn drive the technology planning to support the business. Technology planning leads to the discovery of further opportunities for future uses of technology, which will influence further business planning and strategy.

Figure 1.2: Strategic alignment cycle

Parker and Benson also recognised the fact that this cyclic process may not work on an organisation-wide scale. Organisations usually do not operate in a way that supports a 'monolithic' view of their information systems. However, they also argue that these cycles can be specialised to a specific line-of-business, or a specific business unit. In other words, the cycle of Figure 1.2 can be applied to smaller, more focussed, scopes within an organisation. Scopes that may range from an entire value-chain, via business-units to teams and individual work places.

The views of Parker and Benson were refined further by Henderson and Venkatraman [HV93]. On the importance of alignment between business and information technology *strategies*, they argue:

> *We argue that the inability to realise value from information technology investments is, in part, due to the lack of alignment between business and information technology strategies of organisations.*

They also conclude:

> *Strategic alignment is not an event, but a process of continuous adaptation and change.*

Alignment, however, does not only play a role at a strategic level. Proper alignment between information systems and their organisational (and human!) context is just as important at a tactical and operational level. For example, at a tactical level one would be concerned with the selection of a specific portfolio of information systems to be developed on the shorter term and how they align to the organisational activities & goals on the shorter term. At the operational level, one would be concerned with the development of a specific information system and its direct organisational and human environment.

1.3.3 Architecture-driven information system development

This textbook takes the view that the use of *architecture* enables the alignment between an information system and its organisational and human context. An alignment that should range from the strategic levels to operational levels. We therefore take the perspective that proper alignment is at the heart of what we define to be *architecture-driven information systems development*:

> **Architecture-driven information system development –** information system development, using architecture as a means to
>
> - guide & control the design and evolution of the information system,
> - evaluate & compare different information system alternatives,
> - negotiate the concerns of the information system's stakeholders,
>
> with the aim of optimising the alignment of the resulting information system to its relevant, technological, organisation and human context (systems).

This definition is a further elaboration of the definitions provided in [Pro98, PBHJ00]. As mentioned before, this text-book mainly focuses on the definition and design processes as these are influenced the most by the use of architecture.

1.4 Structuring the domain

The aim of this text-book is to discuss several aspects from the domain of architecture-driven information system development. Even though the aim of this book is *not* to define a specific method for architecture-driven information system development, it makes sense to use a methodological framework to structure the contents of this text-book.

1.4.1 Methodological framework

At the end of the eighties of the last century, there was a growing need to compare different information system development methods. For example, in [SWS89, WH90, HW92] different frameworks and strategies are discussed to position development methods. Approaches to chop-down the so-called *methodology jungle*.

The methodological framework we will employ, is based on the framework originally presented in [WH90] and [SWS89], and refined further, with a way of communicating, in [PW94, PW95b] and [Pro94]. In the resulting framework, a modelling method is dissected into the following six aspects:

Way of thinking – articulates the assumptions on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* [Sol83, WAA85], *underlying perspective* [Mat81] or *philosophy* [Avi95].

Way of modelling – identifies the *core concepts* of the language that may be used to denote, analyse, visualise and/or animate system descriptions.

Way of communicating – describes how the abstract concepts from the way of modelling are communicated to human beings, for example in terms of a textual or a graphical notation.

The way of communicating essentially forms the bridge between the way of modelling and the way of working, it matches the abstract concepts of the way of modelling to the pragmatic needs of the way of working.

Way of working – structures (parts of) the way in which a system is developed. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.

Way of controlling – the managerial aspects of system development. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management and governance (see [Ken84] and [Sol88]).

Way of supporting – the support to system development that is offered by (possibly automated) tools. In general, a way of supporting is supplied in the form of some computerised tool (see for instance [McC89]).

Figure 1.3: Aspects of a method

The arrows in 1.3 should be interpreted as: if $x \rightarrow y$, then aspect x supports aspect y. The combination of a way of modelling and communicating is usually referred to as a "modelling technique".

Some methods may provide a very detailed way of working. For example, for the information modelling method as discussed in [Hal01], a very detailed way of working is presented. However:

A method should never become an excuse to stop thinking!

This text-book primarily focusses on a way of thinking, way of modelling and provides some aspects of a way of working and way of controlling for architecture-driven information system development.

For as far as this text-book portrays a complete *method* for the definition and design parts of architecture-driven development of information systems, its way of thinking is that the actual ways of modelling, working, controlling and support should be highly situational. In [WAA85], the authors state:

> *In a practical discipline, one must always distinguish between the* **ideal methodology** *as thought in text-books, and the realities of any situation which causes departure from the ideal in order to allow for the exigencies of the real world. Many design methodologies are prescriptive not only of what must be done but of the order in which it has to be done. In the real world, decisions are often made before all the facts have been gathered.*

In other words, let the reader be warned. The document you are currently reading is a text-book. A text-book that should inspire. It, however, will not provide all-inclusive answers that will fit all practical situations.

1.4.2 Information systems are systems too

Information systems are systems! To be more precise, they are generally systems that, in addition to processing information, are:

- capable of undergoing (state) changes,
- able to perform actions,
- able to respond to external triggers,

in other words, they are so-called [FVSV+98] open and active systems. The latter three properties hold for numerous other systems as well. Some random examples include:

- The human nervous system.
- An ant colony.
- A train.
- A school of fish.
- A group of people.

The list is, obviously, sheer endless. When we discuss information systems and their design, we consider it to be worthwhile to first look at systems from a more general perspective.

In civil and military architecture it has become an accepted practice to copy patterns from constructions in nature and use them in our own constructions. In line with this practice, it makes perfect sense to see if we can use patterns and properties of other systems, such as systems that occur in nature, in the design of information systems. For instance, the way in which the development and design of software agents is approached, draws more and more on the way biological organisms interact and grow [Ode00a, Ode00b, WJK00]. In the field of complex-adaptive systems [HM95], even more inspiration may be gained on the development of information systems that are better equipped to deal with complexity and evolution of their environment.

Rechtin and Maier in [Rec91, MR02], also subscribe to the point of view that it is worthwhile to look at architecture-driven development of systems in general as it allows the development and evolution of specific classes of systems to benefit from each other's insights.

In these observations lies our motivation to first look at systems in general before zooming in on information systems in particular.

1.4.3 Structure of this text-book

This text-book is split into four themes:

- Architecture
- Modelling
- Definition
- Design

For each of these themes, a seperate chapter discusses the themes from a general systems perspective as well as an information systems perspective.

Questions

1. What is an information system? Give some examples of an information system.
2. What is information system development? What does 'architecture-driven' add to this?
3. What are the consequences of an evolving environment on information systems?
4. Sometimes organisations use the introduction of IT as a way to force changes in organisations. Why is this likely to fail?
5. Why is complexity a challenge for information system development?

   What will happen if you add evolution to the equation?
6. Why is it important to use an 'architecture-driven' approach for the development of systems that are used by some organisation in achieving its goals?

   Which role is played by the environment of the organisation?

   What is the role of the possible evolution of the organisation and its environment?
7. What is the essence of architecture in an information systems context?
8. Explain, in your own words, the essence of alignment.
9. Why is it important to optimise the alignment of an information system to its context?

Recommended Reading

1. J. Odell. Agents (part 2): Complex systems. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

   This report discusses how the complex adaptive systems (or simply, complex systems) approach is applicable to developing multiagent applications. In particular, it discusses topics such as adaptation, emergence, "edge-of-chaos" phenomena, and the difference between agents and objects.

   This makes it a useful introduction to the promise of using principles and patterns from the structure of other types of systems for the design of (computerised) information systems.
2. E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

   Chapter 1 in particular.

3. A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0-632-01216-8

   This book offers an inspiring view on information system development. Even though the book may be regarded as dated, it should still provide ample inspiration to people who are new to the field of information system development.

   The book presents a multi-view approach to the development of information systems, distinguishing five points of view – organisation's human activities, information, socio-technical, human-computer and technical.

Optional Reading

1. B.H. Boar. *Practical steps for aligning information technology with business strategies*. Wiley, New York, New York, 1999. ISBN 0471076376
2. M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL.* Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858
3. E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, 1998. ISBN 3-901-88201-4
4. J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4–16, 1993.
5. P.W.G. Keen. *Shaping the Future - Business Design Through Information Technology*. Harvard Business School Press, Boston, Massachusetts, USA, 1991. ISBN 0875842372
6. M.M. Parker and R.J. Benson. Enterprisewide information management: State-of-the-art strategic planning. *Journal of Information Systems Management*, (Summer):14–23, 1989.
7. D. Tapscott and A. Caston. *Paradigm Shift – The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993. ASIN 0-070-62857-2

Bibliography

[Avi95] D.E. Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, USA, 2nd edition, 1995. ISBN 0077092333

[BCK98] L. Bass, P.C. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, Massachusetts, USA, 1998. ISBN 0-201-19930-0

[Boa99] B.H. Boar. *Practical steps for aligning information technology with business strategies*. Wiley, New York, New York, 1999. ISBN 0471076376

[BP88] B.W. Boehm and P.N. Papaccio. Understanding and controlling software costs. *IEEE Transactions of Software Engineering*, 14(10):1462–1477, October 1988.

[Bro95] F. Brooks. *The Mythical Man-Month; anniversary edition.* Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201835959

[Coc01] S. Cochran. The rising cost of software complexity. *Dr. Dobb's Journal*, April 2001.

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL.* Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, 1998. ISBN 3-901-88201-4

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1-55860-672-6

[Ham90] M. Hammer. Re-engineering work: don't automate, obliterate. *Harvard Business Review*, 68(4):104–112, April 1990.

[HM95] J.H. Holland and H. Mimnaugh, editors. *Hidden Order : How Adaptation Builds Complexity*. Perseus Press, Cambridge, Massachusetts, 1995. ISBN 0201442302

[HV93] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4–16, 1993.

[HW92] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, September 2000. ISBN 0-738-12518-0
`http://www.ieee.org`

[Kee91] P.W.G. Keen. *Shaping the Future - Business Design Through Information Technology*. Harvard Business School Press, Boston, Massachusetts, USA, 1991. ISBN 0875842372

[Ken84] F. Kensing. Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon-DeMarco Approach. In T.M.A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 325–338. North-Holland, Amsterdam, The Netherlands, 1984.

[LS80] B. Lientz and E. Swanson. *Software Maintenance Management – a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley, Reading, Massachusetts, 1980. ISBN 0201042053

[Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. (In Danish).

[McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003.
`http://www.webster.com`

[MR02] M.W. Maier and R. Rechtin. *The Art of System Architecting*. CRC Press, Boca Raton, Florida, 2nd edition, 2002. ISBN 0849304407

[NP90] J. Nosek and P. Palvia. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance*, 3(2):157–174, 1990.

[Ode00a] J. Odell. Agents (part 1): Technology and usage. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[Ode00b] J. Odell. Agents (part 2): Complex systems. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[PB89] M.M. Parker and R.J. Benson. Enterprisewide information management: State-of-the-art strategic planning. *Journal of Information Systems Management*, (Summer):14–23, 1989.

[PBHJ00] H.A. Proper, H. Bosma, S.J.B.A. Hoppenbrouwers, and R.D.T. Janssen. An Alignment Perspective on Architecture-driven Information Systems Engineering. In D.B.B. Rijsenbrij, editor, *Proceedings of the Second National Architecture Congres*, Amsterdam, The Netherlands, EU, November 2000.

[Pro94] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 90-9006849-X

[Pro98] H.A. Proper. Da Vinci – Architecture-Driven Business Solutions. Technical report, Origin, Amsterdam, The Netherlands, EU, Summer 1998.

[Pro01] H.A. Proper, editor. *ISP for Large-scale Migrations*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU, 2001. ISBN 9076304882

[PW94] H.A. Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.

[PW95a] H.A. Proper and Th.P. van der Weide. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.

[PW95b] H.A. Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0-444-86705-8

[Sol88] H.G. Sol. Information Systems Development: A Problem Solving Approach. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.

[SWS89] P.S. Seligmann, G.M. Wijers, and H.G. Sol. Analyzing the structure of I.S. methodologies, an alternative approach. In R. Maes, editor, *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, The Netherlands, EU, 1989.

[Tap96] D. Tapscott. *Digital Economy - Promise and peril in the age of networked intelligence*. McGraw-Hill, New York, New York, USA, 1996. ISBN 0070633428

[TC93] D. Tapscott and A. Caston. *Paradigm Shift – The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993. ASIN 0-070-62857-2

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0-632-01216-8

[WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, EU, 1990. Springer-Verlag, Berlin, Germany, EU. ISBN 3-540-52625-0

[WJK00] M. Wooldridge, N.R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

Chapter 2

Systems and their Architecture

Version:
17-07-2002

2.1 System concepts

2.1.1 Introduction

We refer to all sorts of things as 'systems'. The broadness of our understanding of the concept of 'system' comes, for example, to the fore in the definition as it may be found in [Mer03]:

> A regularly interacting or interdependent group of items forming a unified whole: as
>
> 1. a group of interacting bodies under the influence of related forces,
> 2. an assemblage of substances that is in or tends to equilibrium,
> 3. a group of body organs that together perform one or more vital functions,
> 4. the body considered as a functional unit,
> 5. a group of related natural objects or forces,
> 6. a group of devices or artificial objects or an organisation forming a network especially for distributing something or serving a common purpose,
> 7. a major division of rocks usually larger than a series and including all formed during a period or era,
> 8. a form of social, economic, or political organization or practice.

The IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [IEE00] takes a function-oriented perspective when defining systems:

> A collection of components organised to accomplish a specific function or a set of functions

In practice, most people intuitively agree on such simple definitions of systems. Apparently these definitions are broad enough to cover the meaning of usual linguistic constructs where 'system' is used. However, when looking at what is regarded as a system in practice, it becomes apparent that some very important aspects of the system concept are missing from the traditional definitions.

In [FVSV+98] some examples are given of what we would, and would not, observe to be systems in our daily life:

Example 2.1.1

One can regard an organisation or a bicycle as systems. Also a Hitchcock film recorded on a video cassette, which is inserted in a video cassette player, which again is connected to a TV-set, could easily be interpreted as a system. Nothing is unusual with such system views, and they are well covered by the definitions. But if you buy some eggs from a farmer and use two of them for breakfast, then the domain of obviously interrelated phenomena: You, the farmer, the farmers hen that laid the eggs, the frying pan you used to prepare the eggs, and the two eggs now in your stomach (and thereby in some transformed form a part of yourself) – this domain might probably not be regarded as a system, because it might be difficult to see a purpose for that. But it fits the definitions.

Or consider a single raindrop in an April shower: It consists of a vast number of water molecules, kept together by surface tension and constantly moving around among each other in a complicated manner controlled by a set of (thermo-) dynamic forces. Again according to the simple definitions above, the drop qualifies as a system. But that is strange, because when you on your way back from the farmer, happen to get soaked in the shower, you might feel it is caused by raindrops – not by systems.

On the other hand, a meteorologist studying possible weather situations that could cause rain, may see a purpose in regarding a raindrop as a system in interaction with the surrounding atmosphere, but in most other situations a raindrop is just a raindrop.

2.1.2 Core concepts

Key to understanding the system concept is to realise that a system is a subjective phenomenon. In other words, it is not an absolute or objective thing. Systems are not a priori given. As Checkland expresses it, there must be a describer/observer who conceives or thinks about a part of the world as a system [Che81]. In other words, it is important, that there is a *system viewer* who can see a purpose in regarding *something* as a system. This purpose should be expressed as at least one meaningful relationship between the domain of elements considered as a whole and the environment.

In defining precisely what we mean by systems, we first need to introduce some core concepts, most of which are based on the ones found in [FVSV+98]. We start be identifying that part of what we perceive to be 'the real world' which we are interested in:

Universe – the 'world' under consideration.

It is presumed that the world consists of elements and relationships between these elements.

Domain – any 'part' or 'aspect' of the universe.

Environment – the environment of a domain is that part of the universe, which influences that domain, or vice versa.

Note that an environment is indeed a domain itself.

Figure 2.1 provides an illustration of these concepts.

The next step in defining systems is the introduction of the *system viewer*, and its conception of *domain* as a system:

Viewer – a human actor perceiving and conceiving (part of) a domain.

Perception – that what results, in the mind of a viewer, when they observe a domain with their senses, and forms a specific pattern of visual, auditory or other sensations in their minds.

Conception – that what results, in the mind of a viewer, when they interpret a perception of a domain.

Figure 2.1: Domains, Environments and the Universe

Note that a viewer may also be considered at an aggregated level. For example, a *single* business manager, observing a business, is indeed a viewer, but the *collective* business management can be seen as a viewer of the business as well.

The conception as it is harboured by a viewer is impossible to communicate and discuss with other viewers, unless this conception is articulated somehow. In other words, it needs to be *described* somehow. Furthermore, in practice, we are not interested in all types of conceptions. Our interest is limited to those conceptions that are precise and explicit enough to be referred to as a *model*:

Description – the result of a viewer denoting a conception, using some language to express themselves.

Model – a purposely abstracted, clear, precise and unambiguous conception (of some domain).

The resulting situation is illustrated in figure 2.2.

In [FVSV$^+$98] the term *representation* is used rather than the term *definitions* as used here. We have chosen to favour the term *description*, as it is the term of choice of [IEE00].

The underlying relationships between viewers, domains, conceptions and descriptions are expressed in terms of the so-called FRISCO tetrahedron [FVSV$^+$98], as depicted in figure 2.3.

2.1.3 Systems

Using these general definitions, we can, in line with [FVSV$^+$98], more precisely define the way we view systems:

System domain – a domain that is conceived to be a system, by some viewer, by the distinction from its environment, by its coherence, and because of its systemic properties.

Figure 2.2: A system viewer, having a conception of a system domain, and describing this in a description.

Figure 2.3: The FRISCO tetrahedron

Systemic property – a meaningful relationship that exists between the domain of elements considered as a whole, the system domain and its environment.

System viewer – a viewer of a system domain.

System – a special model of a system domain, whereby all the things contained in that model are transitively coherent, i.e. all of them are directly or indirectly related to each other and form a coherent whole.

A system is conceived as having assigned to it, as a whole, a specific characterisation (a non-empty set of systemic properties) which, in general, cannot be attributed exclusively to any of its components.

To distinguish between colloquial use of the term system and the more formal use of the fact that it is actually a model, we will use the term system model, whenever we need to stress the fact that we refer to the fact that it actually *is* a model.

System description – the description of a system.

As the definition states, in general, the systemic properties of a system cannot be attributed exclusively to any of its constituent components. For example, none of the constituent parts of a train has the exclusive "train property". Together, however, the parts do have the "train property". In other words, the whole is more than just the collection of its parts. The farmer-you-frying-pan-eggs-hen situation as discussed in the above example, is a situation which may not constitute a 'whole' with any sensible systemic property. In which case we will not consider it to be a system.

As identified in [FVSV$^+$98], there is a potential objection against the above subjectivity-based definition of system. In daily life, it is quite sensible to talk about "designing, constructing and implementing a system" or "to interact with a system". The use of the terms 'system' gives associations to this term as denoting something that can be interacted with in a rather concrete way and not just as a conception. These associations, however, do not lead to any inconsistencies. These example phrases are simply convenient abbreviations for more elaborate expressions. For instance, "to interact with a system" really means:

to interact with phenomena in the system domain that is conceived as a system (because of its systemic properties).

To "design, construct and implement" a system really means:

to bring together and structure phenomena in a particular part of the world (which then becomes the system domain) with the purpose of constructing them such they together have certain systemic properties.

It is also interesting to note that the word system originates from the Greek word "Syn-histanai", which means: *to cause to stand.*

As mentioned before, our main interest lies with open and active systems. We are now in a position to more precisely define these types of systems more specifically:

Active system – a special kind of system that is conceived of as begin able to change parts of the universe.

Dynamic system – a special kind of system that is conceived of as undergoing change in the cause of time.

Open system – a special kind of dynamic system that is conceived as reacting to external triggers, i.e. there may be changes inside the system due to external causes originating from the system's environment.

Note that a system may be active and yet be non-dynamic. For example, the mere presence of a dummy speeding camera, i.e. one that is able to capture speeding vehicles on film, may lead drivers to drive more slowly. The dummy speeding camera may thus be seen as an active, yet non-dynamic, system.

Models, such as systems, of domains are presumed to consist of elements. In other words, we define:

Model element – any part of a model.
The set of elements in a model is represented formally by the set $\mathcal{MEL}$.

A system may now be identified as a set of elements, in other words, if S is a system, then $S \subseteq \mathcal{MEL}$. The model of the environment of a system S may be defined as a set $E \subseteq \mathcal{MEL}$, where $S \cap E = \varnothing$.

When considering models of domains, we may actually distinguish between two specific classes of elements:

Model relationship – any model element that is conceived as a relationship between entities from the model.
Formally, the set of model relationships is represented as $\mathcal{MRL}$, where $\mathcal{MRL} \subseteq \mathcal{MEL}$.

Model entity – any model element that is not conceived of as a model relationship.
The set of model entities is formally represented by $\mathcal{MEN} \triangleq \mathcal{MEL} - \mathcal{MRL}$.

The distinction between a relationship and an entity for a given model, may not always be that clear. In other words, the distinction is rather subjective. It all depends, to no surprise, on the viewer of a domain.

The model relationships are presumed to run between model entities. In other words, we presume to exist two functions:

$$\mathit{From} : \mathcal{MRL} \rightarrow \mathcal{MEN} \quad \text{and} \quad \mathit{To} : \mathcal{MRL} \rightarrow \mathcal{MEN}$$

yielding the source and destination entities of a relationship respectively.

As an abbreviation, we also introduce the function $\mathit{Involved} : \mathcal{MRL} \rightarrow \wp(\mathcal{MEN})$, which is defined as:

$$\mathit{Involved}(r) \triangleq \{\mathit{From}(r), \mathit{To}(r)\}$$

Note that if S is a system and E is a model of its environment, then we must have:

$$\forall_{e \in E}\, \exists_{r \in \mathcal{MRL}, s \in S}\, [e, s \in \mathit{Involved}(r)]$$

2.1.4 Sub-systems

To gain a better understanding of complex systems it has proven to be useful to identify smaller-scale systems within a larger system, leading to sub-systems. A detailed discussion on dealing with complexity by systems in general, and the role played by hierarchical decomposition, may be found in e.g. [Sim62].

A sub-system may, in line with [FVSV+98], defined as:

Sub-system – a sub-system S' of a system S, is a system where the set of model elements in S' is a subset of the elements in S. Formally, since we identify systems by means of their underlying set of elements, this is defined as: $S \subseteq S'$.

Two common dimensions along which to define sub-systems are: component-systems and aspect-systems.

Component-system – a component-system S' of a system S, is a sub-system, where the set of model entities in S' is a proper subset of the set of entities in S. This is captured formally by means of the relation $\sqsubset_c \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$, where:

$$S' \sqsubset_c S \triangleq S' \subseteq S \wedge (S' \cap \mathcal{MEN}) \subset S$$

Aspect-system – an aspect-system S' of a system S, is a sub-system, where the set of model relationships in S' is a proper subset of the set of relationships in S. This is captured formally by means of the relation $\sqsubset_a \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$, where:

$$S' \sqsubset_a S \triangleq S' \subseteq S \wedge (S' \cap \mathcal{MRL}) \subset S$$

Note that some authors, for example [Vel92, Bem98], use the term "sub system" to refer to the above defined concept of component-system. However, we prefer to use the term sub-system as defined above, as it allows us to view it as a generalisation of the concepts component-system and aspect-system.

Different viewers may disagree on the fact whether some sub-system is an aspect-system or a component-system (or a combination thereof). This can be traced back to the subjectivity involved in distinguishing between model relationships and model entities. Whenever there is a 'clear' analogy to physical structures, it will be easier to identify the difference. Consider a freight-train as an example system. Typical component-systems of such a system are: the locomotive, the engine-driver, several types of box-cars, etc. An aspect-system of a freight-train would be the hydraulic braking system of the train as a whole.

A sub-system is indeed a system. As such, a sub-system S' of a system S will also have its own systemic properties. However, these properties are most likely no subset of the systemic properties of S. For example, the engine-driver's systemic properties are by no-means a clear subset of the systemic properties of a freight-train.

Note that the sub-system of an open active system does not have to be an open active system itself. In other words, even though our main interest lies with open active systems, we may quite well need to consider non-open or non-active sub-systems of these systems.

Using the above defined notion of sub-system, we are now also able to better understand our definition of organisational system, information system and computerised information system:

Organisational system – a special kind of system, being normally active and open, and comprising the conception of how an organisation is composed and how it operates (i.e. performing specific actions in pursuit of organisational goals, guided by organisational rules and informed by internal and external communication), where its systemic properties are that it responds to (certain kinds of) changes caused by the system environment and, itself, causes (certain kinds of) changes in the system environment.

Information system – a sub-system of an organisational system, comprising the conception of how the communication and information-oriented aspects of an organisation are composed and how these operate, thus leading to a description of the (explicit and/or implicit) communication-oriented and information-providing actions and arrangements existing within the organisational system.

Computerised information system – a sub-system of an information system, whereby all activities within that sub-system are performed by one or several computer(s).

In chapter **??**, we will provide even more background to these definitions.

2.1.5 System views

The same set of elements that comprise a system domain may be regarded by different system viewers. This is bound to lead to different system models, depending on the specific viewers. The fact that when referring to the same system domain, people are likely to refer to different system models, is one serious cause for the current confusion in our professional domain. People, tend to think about a system as something that can be objectively determined [FVSV$^+$98].

One of the key distinguishing factors in viewing system domains is the particular interest which a viewer has when observing a system domain. Consider, as an example, the domain of a freight-train and the course it takes on a railway network. One person, for example the owner of freight on the train, may see it as a useful transport system in action, which is able to move large objects from one location to another in a convenient way. The engine driver alone would not be able to do so, nor could the train by itself, but in combination they can. A shunting yard operator would view this system domain quite differently, being a controllable system which behaviour can be directed by manipulating switches. Again, the engine driver of a be-lated passenger train would probably regard the same freight-train crossing their path as nuisance, causing additional delays. Here we have three views of the same domain, but with quite different sets of systemic properties, leading to different perceptions, conceptions and eventually models of the same system domain.

Even more, the same viewer may indeed be able to distinguish between different systemic properties that may be attributed to the same system domain, leading to different system models. For example, the three views on a freight-train as sketched above, may actually be harboured by a single person. In each case, the set of systemic properties observed by the viewer, with regards to the system domain, depends on their specific interest. In the above case, the interests would be: *means of transportation*, *object to be controlled* and *causes for delay*.

The concept of *system view* may now be defined as follows:

System view – a model of a system domain from the perspective of a related set of interests of a system viewer.

Interest – the specific reason(s) why a system viewer observes a system domain. This is usually a confluence of the systemic properties of interest to the system viewer and the aspects of the system that are considered relevant to these systemic properties.

Each specific view clearly depends on the specific system viewer and the interests of that viewer. Note that not only viewers may take the form of aggregations, but interests as well. For example, the interest *means of transportation* can actually be seen as an aggregation of the following interests:

- what a *means of transportation* is conceptually,
- possible physical realisations of *means of transportation*,
- the evolution over time of different *means of transportation*.

This will obviously lead to aggregated system views, where a view covering some aggregated interest, essentially consists of sub-views covering all sub-interests.

2.1.6 System evolution

As discussed above, a system is identified as a set of elements from $\mathcal{MEL}$. However, as the kind of systems we are mainly interested in, are open and active, the system domain is bound to change over time, as should the model. In other words, a set of elements S from $\mathcal{MEL}$ really refers to a static view of a system, as it is valid at some fixed point of time.

We really should not only take snapshots of systems into considerations, but rather the evolution of a system as a whole. Let us start by introducing some time dimension $\mathcal{TME}$. At the moment we do not yet presume $\mathcal{TME}$ to have an explicit ordering defined over it. Later on, we wil indeed need to introduce some notion of temporal order.

With a time dimension $\mathcal{TME}$, we can view a systems evolution as a set of co-evolving elements. In other words, a systems evolution over time can be regarded as a set element evolutions:

Element evolution – the evolution, over time, of a system element.
The set of system elements is captured formally as: $\mathcal{EEV} \triangleq \mathcal{TME} \rightarrowtail \mathcal{MEL}$.

Note that the $\rightarrowtail$ symbol[1] is used to identify partial functions.

A system's evolution may now be identified as a subset E of $\mathcal{EEV}$. More generally, we introduce:

System evolution – the evolution, over time, of a system.
The set of possible system evolutions is represented formally by the set $\mathcal{SEV} \triangleq \wp(\mathcal{EEV})$.

The different snapshots of the system that are valid at different points of time, are accessible by means of the following function:

$$\_@\_ : \mathcal{SEV} \times \wp(\mathcal{TME}) \rightarrow \wp(\mathcal{MEL})$$

where:

$$E_{@t} \triangleq \{ e(t) \mid e \in E \}$$

Snapshots should be complete with regards to the sources and destinations of relationships:

[SY1] (*temporal conformity*) If $E \in \mathcal{SEV}$, then:

$$\forall_{E \in \mathcal{SEV}} \forall_{r \in \mathcal{MRL} \cap E_{@t}} [From(r) \in E_{@t} \wedge To(r) \in E_{@t}]$$

2.2 System development

Before we can discuss system architecture, we first need to build up some more understanding of the concepts involved in the development of systems.

2.2.1 Key processes

In chapter 1, we defined information system development to consist of four key processes:

Definition process – a process aiming to identify all requirements that should be met by the system and the system description.
In literature this process may also be referred to as requirements engineering.

Design process – a process aiming to design a system conform stated requirements. The resulting system design may range from high-level designs, such as an strategy or an architecture, to the detailed level of programming statements or specific worker tasks.

Construction process – a process aiming to realise and test a system that is regarded as a (possibly artificial) artifact that is not yet in operation.

Installation process – a process aiming to make a system operational, i.e. to implement the use of the system by its prospective users.

We claim that these four processes are key to system development in general.

When integrating these processes into a unified process, a first, and naive, way of representing the resulting overall development process, would lead to the situation as shown in figure 2.4. In the situation depicted, it is presumed that some operational system exists, and that there is a need to change/improve/extend this system. By means of an definition process, the requirements of this desired system change can be ascertained. Using these requirements as a starting point, a new system can be designed, leading to a

Figure 2.4: System development

system design. Based on this design a new system may be constructed, which, after completion, may then be installed as part of the operational system.

This representation of the development process is, however, naive in two important ways:

- It suggests a linear flow of activities.
- It does not distinguish between *system domains*, *conceptions* of *systems* and *system descriptions*.

The arrows in figure 2.4 should indeed not be interpreted as putting a requirement on the *start* of the processes, but rather of the *finalisation* of them. In other words, the processes may quite well run in parallel, however, the definition process should be finalised before the design process can be finalised, etc. There are actually three major flavours of development approaches that may be used [FV99]:

Linear approach – step by step execution of a (part of a) development process, where a consecutive step is not executed until the preceding step is finished.

Incremental approach – a (part of a) development process is executed on a sub-system by sub-system basis, using some well-defined division of a system into sub-systems.

Evolutionary approach – a (part of a) development process is executed completely in several iterations, leading to several consecutive versions of the set of deliverables.

These flavours may actually be mixed/matched for different parts/stages of the development process. For example, the following recipe may be used for the execution of a project:

Do linearly:

1. Do evolutionary:
   - Definition
   - Design

[1] In appendix A, a brief discussion on the mathematical conventions used is provided.

2. Do incrementally for all top-level component-systems:

   Do linearly:

   (a) Construction

   (b) Installation

In chapter **??**, where we will focus on a way of controlling for system development, we will discuss these options in more detail. In the next chapter, where we will discuss system requirements, we will also see how, in general, it is nearly impossible to first elicit all relevant requirements before starting design.

With regards to the distinction between system domain, conceptions of systems and system descriptions, a more refined view of figure 2.4 is given in figure 2.5. Both the operational system (needing a change) and the constructed system (providing the actual change) are parts of the universe, i.e. that what we perceive to be 'the real world'. In other words, both the constructed system and the operational system from figure 2.4 actually refer to system domains in the universe. The system requirements, as well as the system design, are really system *descriptions*.

Figure 2.5: System development – refined view

2.2.2 Stakeholders and their concerns

During the development of a system, different people may have different interests with regards to the system. Collectively, we will refer to these people as stakeholders:

Stakeholder – a party (a system viewer) with a specific interest pertaining to a system's development, its operation or any other aspects that are critical or otherwise important.

Examples are: Users, operators, owners, architects, engineers, testers, project managers, business management, ...

The interests of a stakeholder with respect to some system to be developed, originate from some deeper motivation: stakeholder's goals:

> **Stakeholder goal –** the end toward which effort is directed by a stakeholder, in which the system (of which the stakeholder is indeed a stakeholder) plays a role.
>
> This may pertain to strategic, tactical or operational end. The role of the system may range from passive to active. For example, a financial controller's goal with regards to a future/changed system may be to control (system) development costs, while the goal of users of the system may be to get their job done more efficiently.

The specific interests of a stakeholder are, in line with [IEE00], referred to as stakeholder concerns:

> **Stakeholder concern –** an interest of a stakeholder, resulting from the stakeholder's goals, and the role played by some system.
>
> This usually pertains to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

Usually we will abbreviate "stakeholder concern" to "concern". Each of these stakeholders are obviously system viewers, viewing the system with specific interests; their concerns.

A system (to be developed) is aimed to be beneficial for some group of stakeholders. This benefit is referred to as the system's mission:

> **System mission –** a role, to the benefit of the goals of stakeholders, for which the system is intended.

This leads to the situation as depicted in figure 2.6. A stakeholder will typically have some operational goal in which the system will/should/may play a role. Due to this potential role, the stakeholder is bound to have some concerns with regards to a pre-existing system or a system to be developed. Facing these concerns we find the system mission, in other words, that what the system aims to be for its stakeholders.

Figure 2.6: Meeting stakeholder concerns

2.3 System viewpoints

As discussed before, given some system domain, there are potentially many, many, different possible views.

2.3.1 The need for viewpoints

Given the abundance of potential view, it is sensible, in the context of a system development project, to define some consensus based ways of regarding systems under development. In other words, pre-defined combinations of classes of stakeholders, their typical concerns, as well as a specification of the conventions for constructing and using views of this class. This is what we will refer to as a system viewpoint:

System viewpoint – a specification of the conventions for constructing and using system views.

This involves:

Way of thinking – articulates the assumptions on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* [Sol83, WAA85], *underlying perspective* [Mat81] or *philosophy* [Avi95].

Way of modelling – identifies the *core concepts* of the language that may be used to denote, analyse, visualise and/or animate system descriptions.

Way of communicating – describes how the abstract concepts from the way of modelling are communicated to human beings, for example in terms of a textual or a graphical notation.
The way of communicating essentially forms the bridge between the way of modelling and the way of working, it matches the abstract concepts of the way of modelling to the pragmatic needs of the way of working.

Way of working – structures (parts of) the way in which a system is developed. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.

Way of supporting – the support to system development that is offered by (possibly automated) tools. In general, a way of supporting is supplied in the form of some computerised tool (see for instance [McC89]).

Way of using – identification of heuristics that:

- define situations, classes of stakeholders and interests, for which a given system viewpoint is most suitable,
- provide guidance in tuning a given system viewpoint to specific situations, classes of stakeholders and their interests at hand. For example, in terms of the set of modelling concepts to be used, effective notations, visualisations, etc.

System viewpoints may serve as a means to standardise deliverables produced during system development, as viewpoints provide a well-defined way of modelling, communicating and working. At the same time, however, the heuristics in its way of using allow for carefull selection of the right system viewpoints for specific situations, as well as the tuning of a selected viewpoint to the situation at hand.

Just as systems and system views may be aggregated, system viewpoints may be be aggregated as well. In figure 2.7 we have illustrated this.

The reader should realise that, implicitly, we have already identified three basic viewpoints on systems, with an increasing set of core concepts:

0	Systems consist of elements	$\mathcal{MEL}$
1	Systems consist of entities & relationships	$\mathcal{MEL}, \mathcal{MEN}, \mathcal{MRL}$
2	Systems evolve over time	$\mathcal{MEL}, \mathcal{MEN}, \mathcal{MRL}, \mathcal{TME}, \mathcal{SEV}$

These are essentially system viewpoints, albeit not complete. We have only discussed their way of thinking and modelling. In the remainder of this chapter, we will identify two additional viewpoints, with even more concepts.

Figure 2.7: Aggregations of systems, views and system viewpoints.

2.3.2 Viewpoint dimensions

For a specific class of systems, different system viewpoints may be identified along different axes. For example, for buildings, one may identify:

Zoning plan, blue print, plumbing, electrical wiring, foundation, ...

Some dimensions by which to identify different system viewpoints are:

- Time horizon
- Aspects
- Abstraction
- Scope
- Development life-cycle

Below we will briefly discuss these dimensions in more detail. One may argue that an additional dimension should be *scope*. Scoping, however, does not have anything to do with different viewpoints. Scoping *does* have everything to do with limiting the system domain that is being viewed. Note that we should not confuse limiting our scope, with our natural tendency to consider a 'large' system domain at a higher level of abstraction.

2.3.2.1 Time horizon

When regarding a system under development, we may look at different time-spans with regards to the system's future. Roughly, we may have a longer-term strategic focus, a medium-term tactical focus, and a short-term operational focus. What we exactly mean by longer-term, medium-term and short-term in terms of a number of years is hard to say in general. This will clearly depend on the kind of system.

Using the distinction between a strategic, tactical and operational level, one may define the following system viewpoints [PBHJ00]:

1. The *system(s) strategy* defines the direction in which system(s) are planned to evolve from a longer term (strategic) perspective.

   As, for example, argued by Mintzberg in [Min94]:

   - strategy is a continual, incremental process of setting and resetting organisational direction;
   - strategy should not be elaborate or detailed, because we cannot anticipate the future in detail;
   - strategy is a dialogue rather than a document;
   - strategy and planning should be done by business managers, not "strategic planners".

   Mintzberg [Min94] stresses the fact that a *system(s) strategy* should be seen as a directional design. Furthermore, this high level design should be seen as the subject of a continuous dialogue between business managers, rather than a fully elaborated document. If expressed as a document, it should really be covered with coffee-stains and scribbles with additional ideas, discussions and refinements. In other words, it should be a perpetual discussion document.

2. The *system(s) plan* provides the translation of the system(s) strategy to a more concrete level. It identifies the system(s) needed in the shorter term that are in line with the longer term strategic view. A system(s) plan may identify multiple plateau's that identify different stages in the development of the system(s) portfolio over time, and should be viewed as a *zoning plan* or a *city plan* for geographical development.

3. Finally, the *system(s) blueprint* defines the design of particular (operational) systems. Where the system(s) plan corresponds to a *zoning plan*, a system(s) blueprint can be seen as the pendant of the blueprint of a building.

Syntactically, system descriptions for these different viewpoints may not even differ. Pragmatically, however, they do in the sense that system viewers adhere a different time horizon to the system models resulting from the different viewpoints.

2.3.2.2 Aspect-systems

Given a system, one may identify several aspect-systems. Even more, there are several ways to identify aspect-systems within a larger system [PBHJ00]. The English language, as well as most other languages, contains a class of words called the *interrogatives* [TM86]:

Which, when, how, what, why, where, whose, ...

These words may be used to formulate questions concerning situations, people, or any other phenomenon we may perceive or conceive. In other words, we may use these interrogatives to identify different relevant aspects of a system. By using questions based on the interrogative words, insight may be gained into different aspects of a system, such as:

Actors, timing, processes, functionality, rationale, purpose, locality, structure, ownership, ...

This would lead to the situation as depicted in figure 2.8, which is taken from [PBHJ00].

The so-called Zachman framework, [Zac87, SZ92], is an example of a framework for information systems that is based on the *what*, *how*, *where*, *when*, *who*, *why* interrogatives, leading in their interpretation to information system aspects:

1. Data
2. Function

Figure 2.8: Different perspectives on a system.

3. Network
4. People
5. Time
6. Motivation

It combines these aspects with five classes of stakeholders:

1. Planner
2. Owner
3. Designer
4. Builder
5. Aspect-contractor

leading to the identification of 30 system viewpoints, see figure 2.9.

An alternative framework of system viewpoints for identifying aspect-systems of information systems is the RM-ODP (Reference Model of Open Distributed Processing) [ISO98] reference model. They identify five system viewpoints:

1. Enterprise viewpoint: the purpose, scope and policies for the system
2. Information viewpoint: the semantics of the information and information processing performed
3. Computational viewpoint: distribution through functional decomposition of the system into objects which interact at interfaces
4. Engineering viewpoint: mechanisms and functions required to support distributed interaction between objects in the system

Figure 2.9: The Zachman framework

5. Technology viewpoint: choices of technology in the system

In a way, these viewpoints can be regarded as aggregations of the system viewpoints identified in the Zachman framework

2.3.2.3 Abstraction

System domains may be viewed at differing levels of granularity. In other words the extend to which component-systems are decomposed into smaller component-systems may differ. For example, one may describe a business at a level of granularity at which business processes are identified while their specific details remain hidden, or one may choose to indeed also show the specific details of business processes by decomposing them. In other words, one may chose to abstract away from details, usually in order to obtain a better overview of the overall structures. This leads to viewpoints that allow modellers to view a system domain at different levels of abstraction.

Abstraction mechanisms

Abstraction allows us to:

- view/analyse/design systems at different levels of 'detail'
- specify/study properties that hold for whole classes of elements
- avoid redundancy
- harness complexity

We identify the following main flavours of abstraction:

- Typing.
- Generalisation.
- Encapsulation.
- Implementation.

A word of caution, with regards to abstraction as a means to harness complexity and focus on commonalities, is in order. Abstracting away from details and/or context, may, during system development lead to blind-spots for issues that are indeed relevant. In Frederick P. Brooks, "*The Mythical-Man Month*" [Bro95], we can read:

> D.L. Parnass of Carnegie-Mellon University has proposed a still more radical solution [Par71]. His thesis is that the programmer is most effective if shielded from, rather than exposed to the details of construction of system parts other that his own. This presuposes that all interfaces are completely and precisely defined. While that is definitely sound design, dependence upon its perfect accomplishment is a recipe for disaster. A good information system both exposes interface errors and stimulates their correction.

Abstraction flavours – Typing

- When modelling systems, we usually like to classify the phenomena we see in terms of types
- For example, we would like to see the activities:
  - Erik marks the exams for 'Systems Architecture – 1'
  - John marks the exams for 'Algorithms – 2'

  as instances of the activity type: Marking exams
- Note: Process-Incarnation versus Process-Specification
- Do these abstractions occur in system descriptions?

Abstraction flavours – Generalisation

- When modelling systems, we also like to be able to consider things in more general terms
- Generalisation

  Person 'Erik' **is a generalisation of** Lecturer 'Erik'

  Person **is a generalisation of** Lecturer

  Commuting **is a generalisation of** cycling from home to work
- Specialisation is the inverse relation
- Do we see these abstractions in system descriptions?

Abstraction flavours – Encapsulation

- When system descriptions become large, we prefer to use some form of explicit information hiding
- Enables step-wise refinement when modelling/designing systems

  Warning: conceptual prejudice!
- Part-whole relationships, aggregation, etcetera, are not the same as encapsulation! However, one may indeed regard a system domain at different levels of abstraction (encapsulation)
- Components in CBD, Objects in OO provide encapsulation by nature
- Do we see these abstractions in system descriptions?

Abstraction flavours – Implementation

- When considering a system domain, we prefer to consider the domain at different levels of specificity with regards to its technological implementation
- In other words, we like to abstract from technological implementations
- For example:
  conceptual level, logical level, physical level
- Remember: you'll need some reference model defining the levels of abstraction with regards to technological implementation
- Do we see these abstractions in system descriptions?

2.3.2.4 Scope

At what scope do we consider a system? For example: enterprise-wide, business-unit or individual worker.

This dimension is likely to be combined with a change in abstraction level in terms of encapsulation of details when using a broader scope.

2.3.2.5 Development life-cycle

A final dimension along which different viewpoints may be associated is the development life-cycle. In figure 2.5, two types of system descriptions have been identified: system requirements and system design. Each type of system descriptions can be seen as a specific (aggregated!) viewpoint on a system, its requirements and its design respectively.

These two viewpoints, requirements and design, are actually the only viewpoints we identify with respect to the development life-cycle, where the system requirements focus on *what* system is needed, while the system design specifies *how* this system may be realised. When using a more refined view of the development life-cycle, one may argue that we could also identify system descriptions such as: a high level-design (e.g. an architecture), a detailed-design, a final-design, etc. We argue that these increasing levels of concreteness of the designs can really be regarded as a continual process of zigzagging between the requirements at some level of concreteness and a design meeting these requirements. This is illustrated in figure 2.10. Starting out from some set of requirements r_1, a design d_1 may be produced. Based on r_1 and d_1, a more detailed set of requirements r_2 may then be identified, which serves as the input to a design process leading to a more detailed design d_2.

2.4 System architecture

Thus far, this chapter has not yet discussed the concept of architecture. This section will remedy this. We will argue that an architectural description, being the description of an architecture, is a system description; albeit one with a very special role. In line with [IEE00], we have defined architecture as:

Architecture – the fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its evolution and design.

Figure 2.10: Refinement of designs

This definition supports the view that an architecture is a (special) conception of a system domain. Even more, it is a model of a system domain. This raises the question: *what is so special about an architecture?* This section is concerned with this very question, leading to a pragmatic definition of architecture as a specific (aggregated) interest with its own viewpoints.

2.4.1 The need for architecture

Architectures, being models of systems, can described in terms of system descriptions. In line with, [IEE00], we will refer to such system descriptions as architectural descriptions.

Architectural description – A system description documenting/describing an architecture.

In the context of software architectures, [BCK98] identifies the following uses for architectural descriptions:

- It is a vehicle for communication among stakeholders.

  A software architecture, often depicted graphically, can be communicated with end users, the client, designers, and so on. By developing scenarios of anticipated use, relevant quality aspects can be analyzed and trade-offs can be discussed with various stakeholders.

- It captures early design decisions, both functional aspects as well as quality aspects.

  In a software architecture, the global structure of the system has been decided upon, through the explicit assignment of functionality to components of the architecture. These early design decisions are important since their ramifications are felt in all subsequent phases. It is therefore paramount to assess their quality at the earliest possible moment. By evaluating the architecture, a first and global insight into important quality aspects can be obtained.

- The global structure decided upon in the architecture, also structures further development.

  The work-breakdown structure may be based on the decomposition chosen at this stage, testing may be organized around this same decomposition, and so on.

- It is a transferable abstraction of a system.

  The architecture is a basis for reuse, for example, as it may be conducted in Component-Based Software Development (CBSD). Design decisions are often ordered, from essential to non-essential features. The essential features are captured in the architecture, while the non-essential features can be decided upon at a later stage. The software architecture thus provides a basis for a family of similar systems, a so-called product line.

It is not hard to see that these uses apply to systems in general! In [IEE00], several potential uses for architectural descriptions have been identified as well:

- Expression of a system and its potential evolution.

  In other words, architectural descriptions are a *means* to approach the co-evolution and complexity of organisations, their context and their information systems. Note that we use the word *approach*, we do not necessarily mean *control*.

- Communication among stakeholders.

  Typical goals are: obtaining directions & feedback, negotiating between contradicting concerns, etc.

- Evaluation and comparison of design alternatives.

- Planning, managing and executing development.

- Expression of the persistent characteristics and supporting principles of a system to guide acceptable change.

- Verification of a system implementation's compliance.

These uses for architectural description are a direct result of the fact that by its very nature, an architecture is a high-level design of a system, which really forms the bridge between the requirements and the (detailed) design of a system. It is also not uncommon that during the design of an architecture, additional requirements will pop up as a result of crucial discussions taking place regarding high-level design decisions.

2.4.2 A pragmatic definition of architecture

The above uses for architectural descriptions, all express possible concerns of a community of stakeholders with regards to:

> the fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its evolution and design.

When combined (aggregated), one may state that these concerns represent "an architectural concern" which a stakeholder may have with regards to a system domain. In line with the above discussion, we propose the following more pragmatic definition of architecture, which does not contradict the earlier definition, but rather provides a pragmatic interpretation:

> **Architecture –** a system model of which the system description, the so-called architectural description, is used during system development to:
>
> - express the fundamental organisation of the system domain in terms of components, their relationships to each other and to the environment and
> - the principles guiding its evolution and design,
>
> and which's explicit intend is to be used as a means:

- of communication & negotiation among stakeholders,
- to evaluate and compare design alternatives,
- to plan, manage, and execute further system development,
- to verify the compliance of a system implementation's.

The pragmatism of this definition lies in the fact that defines architecture in terms of *what you do with it*. This makes architecture really into a special role that may be played by system descriptions.

As a consequence, we can define specific types of architectures in terms of the viewpoints used for system descriptions being used as architectures. If V is the name of some viewpoint and in some system development process system descriptions conform this viewpoint are used in the role of an architecture, we can refer to these architectures as V-architectures. For example, consider a viewpoint that allows us to make high level descriptions of information systems. If we use this viewpoint to describe architectures for information systems, we may refer to the resulting architectures and architectural descriptions as information architectures. This may lead to several classes of architecture, such as:

> Enterprise architecture, software architecture, information architecture, application architecture, security architecture, process architecture, etc.

The question of which architectures to use during the development of a specific system is similar to the question of which viewpoints to use. There is no silver-bullet answer possible; it is highly situationally determined. Let it therefore also be clear, that what one organisation would use as an architecture, another organisation might not use in the role of an architecture, but rather in the role of a sketch, moving towards a more detailed design which they would use as an architecture. In other words:

> *What an architecture is or isn't, is in the role it plays. Don't look for it in its contents!*

2.4.3 Architecture-driven system development

In the previous chapter we defined the notion of architecture-driven information system development, we can now generalise to systems in general as:

> **Architecture-driven system development –** system development, using architecture as a means to
>
> - guide & control the design and evolution of the system,
> - evaluate & compare different system alternatives,
> - negotiate the concerns of the system's stakeholders,
>
> with the aim of optimising the alignment of the resulting system with its relevant, technological, organisational and human context (systems).

Questions

1. Why is it important to realise that system development is not necessarily a linear process?
2. Name three different axes along which a high-level design may be 'refined' to a more detailed design.
3. Describe, in your own words, the relationships between the concepts of: domain, viewer, conception, perception and architecture.
4. What is the difference between: an architecture, architectural description, architectural view, and architectural viewpoint?

5. Why is it important to carefully select relevant system viewpoints when developing systems?

6. Make a meta-model of the concepts introduced by the IEEE recommended practice for architecture [IEE00]. Make sure to include as many constraints as you can deduce from the text.

7. Why is it important to acknowledge the fact that different stakeholders will have different views on a pre-existing or a future system?

8. Make a meta-model of the concepts introduced with respect to systems, domains, viewers, etc.

9. Identify, for the system description language you have see so-far as part of your studies:

   (a) The main concepts of these languages.

   (b) Typical interests and viewers for which these languages may be useful.

10. Why should the set of viewpoints that will be used to describe different aspects of a system that is being developed, be selected carefully. x

11. What makes us consider a system description to be an architecture description?

12. What are possible dimensions for refinements of system descriptions?

13. Why is it important to consider responsibilities of system entities and collaborations among them?

14. Consider a candy vending machine and people purchasing candy from the machine. What would, in such a domain, be the relevant system entities? What would be there responsibilities and collaborations?

15. Suppose you would design a pocket calculator. What would be the essential system elements (at a functional level)? Maybe you could do a role-playing game with a group

Recommended Reading

1. Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, September 2000. ISBN 0-738-12518-0
`http://www.ieee.org`

2. E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

3. J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

Optional Reading

1. M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

2. E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, 1998. ISBN 3-901-88201-4

Bibliography

[Avi95] D.E. Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, USA, 2nd edition, 1995. ISBN 0077092333

[BCK98] L. Bass, P.C. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, Massachusetts, USA, 1998. ISBN 0-201-19930-0

[Bem98] T.M.A. Bemelmans. *Bestuurlijke Informatiesystemen en Automatisering*. Kluwer, Deventer, The Netherlands, 7th edition, 1998. In Dutch. ISBN 9026727984

[Bro95] F. Brooks. *The Mythical Man-Month; anniversary edition*. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201835959

[Che81] P. Checkland. *Systems thinking, systems practice*. John Wiley & Sons, New York, New York, USA, 1981. ISBN 0-471-27911-0

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, 1998. ISBN 3-901-88201-4

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, September 2000. ISBN 0-738-12518-0
`http://www.ieee.org`

[ISO98] *Information technology – Open Distributed Processing – Reference model: Overview*, 1998. ISO/IEC 10746-1:1998(E).
`http://www.iso.org`

[Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. (In Danish).

[McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003.
`http://www.webster.com`

[Min94] H. Mintzberg. *The Rise and Fall of Strategic Planning*. Free Press, New York, New York, 1994. ISBN 0029216052

[Par71] D.L. Parnas. Information distribution aspects of design methodology. In C.V. Freiman, J.E. Griffith, and J.L. Rosenfeld, editors, *Information Processing 71, Proceedings of IFIP Congress 71*, volume 1 – Foundations and Systems, Ljubljana, Yugoslavia, August 1971. North-Holland. ISBN 0720420636

[PBHJ00] H.A. Proper, H. Bosma, S.J.B.A. Hoppenbrouwers, and R.D.T. Janssen. An Alignment Perspective on Architecture-driven Information Systems Engineering. In D.B.B. Rijsenbrij, editor, *Proceedings of the Second National Architecture Congres*, Amsterdam, The Netherlands, EU, November 2000.

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[Sim62] H.A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, volume 106, pages 467–482, 1962.

[Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0-444-86705-8

[SZ92] J.F. Sowa and J.A. Zachman. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3):590–616, 1992.

[TM86] A.J. Thomson and A.V. Martinet. *A Practical English Grammar*. Oxford University Press, Oxford, United Kingdom, fourth edition, 1986. ISBN 3810905798

[Vel92] J. in 't Veld. *Analyse van organisatieproblemen – Een toepassing van denken in systemen en processen*. Stenfert Kroese, Leiden, The Netherlands, 1992. In Dutch. ISBN 9020722816

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0-632-01216-8

[Zac87] J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

Chapter 3

Systems Modelling

Version:
17-07-2002

This chapter aims to discus system modelling in general, irrespective of whether we deal with requirements or design models.

3.1 Refinements between system views

Refinements

- One system view may be more general than another one
- If S and S' are system views, in other words system models, then we may have:

$$S \,\mathsf{RefinesTo}\, S'$$

  S is more *what* than S'
- Further on, we will see that designing a system, really involves a process of constant refinement of system views

Refinements

- $S \,\underline{\mathsf{RefinesTo}}\, S' \;\triangleq\; S \,\mathsf{RefinesTo}\, S' \vee S = S'$
- This may be generalised to evolutions:

$$E \,\underline{\mathsf{RefinesTo}}\, E' \;\triangleq\; \forall_t \,[E_{@t} \,\underline{\mathsf{RefinesTo}}\, E'_{@t}]$$

- From which we can derive:

$$E \,\mathsf{RefinesTo}\, E' \;\triangleq\; E \,\underline{\mathsf{RefinesTo}}\, E' \wedge \neg(E' \,\underline{\mathsf{RefinesTo}}\, E)$$

3.2 Abstraction mechanisms

Abstraction constitutes a special class of relationships in a system model. In other words, we can identify a set of abstraction relationships $\mathcal{ABS} \subseteq \mathcal{MRL}$.

3.2.1 General properties of abstractions

The set of $\mathcal{ABS}$ relationships should not be self-referring:

[SY2] (*non self-referring*) $\forall_{a \in \mathcal{ABS}} [Involved(a) \cap \mathcal{ABS} = \varnothing]$

Using this set of relationships, we can, for a given set of system elements, define the following abbreviations:

$$\begin{aligned} x \,\mathsf{AbstractionOf}_S\, y &\triangleq \exists_{r \in (S \cap \mathcal{ABS})} [x = From(r) \wedge y = To(r)] \\ x \,\mathsf{AbstractionOf}\, y &\triangleq x \,\mathsf{AbstractionOf}_{\mathcal{MEL}}\, y \end{aligned}$$

We presume AbstractionOf to be transitive and irreflexive:

[SY3] (*transitivity*) $x \,\mathsf{AbstractionOf}\, y \,\mathsf{AbstractionOf}\, z \Rightarrow x \,\mathsf{AbstractionOf}\, z$

[SY4] (*irreflexive*) $\neg(x \,\mathsf{AbstractionOf}\, x)$.

We furthermore should have:

[SY5] (*entity conformity*) $x \,\mathsf{AbstractionOf}\, y \wedge y \in \mathcal{MEN} \Rightarrow x \in \mathcal{MEN}$

[SY6] (*relationship conformity – 1*) If $x \,\mathsf{AbstractionOf}\, y$ and $y \in \mathcal{MRL}$, then:

$$x \in \mathcal{MEN} \Rightarrow x \,\mathsf{AbstractionOf}\, From(y) \wedge x \,\mathsf{AbstractionOf}\, To(y)$$

[SY7] (*relationship conformity – 2*) If $x \,\mathsf{AbstractionOf}\, y$ and $y \in \mathcal{MRL}$, then:

$$x \in \mathcal{MRL} \Rightarrow From(x) \,\mathsf{AbstractionOf}\, From(y) \wedge To(x) \,\mathsf{AbstractionOf}\, To(y)$$

Note that for some specific x and y, it is indeed allowed to have:

$$x \,\mathsf{AbstractionOf}\, y \wedge y \in \mathcal{MRL} \wedge x \in \mathcal{MEN}$$

The AbstractionOf relation may be generalised to systems as follows. Let $S, S' \subseteq \mathcal{MEL}$ be systems, then:

$$S \,\mathsf{AbstractionOf}\, S' \triangleq S \subset S' \wedge \forall_{y \in (S' - S) - \mathcal{ABS}} \exists_{x \in S} [x \,\mathsf{AbstractionOf}_{S'}\, y]$$

The RefinesTo and AbstractionOf relations should honour each other:

[SY8] (*sound abstraction*) If $S, S' \subseteq \mathcal{MEL}$, then: $S \,\mathsf{AbstractionOf}\, S' \Rightarrow S \,\mathsf{RefinesTo}\, S'$

[SY9] (*complete abstraction*) If $S, S' \subseteq \mathcal{MEL}$, then: $S \,\mathsf{RefinesTo}\, S' \Rightarrow S \,\mathsf{AbstractionOf}\, S'$

3.2.2 Typing

To be done.

Specialisation of AbstractionOf: TypeOf.

3.2.3 Generalisation

To be done.

Specialisation of AbstractionOf: Generalises.

Typing & Generalisation:

$$x \,\mathsf{Generalises}\, y \wedge y \,\mathsf{TypeOf}\, z \Rightarrow x \,\mathsf{TypeOf}\, z$$

3.2.4 Encapsulation

To be done.

3.2.5 Implementation

In the design of a system, it is quite common to first come up with a description of the system at a high level of abstraction, abstracting away from as many technological considerations as practically possible. The highest of such levels has, at least in the information systems field, generally been referred to as the conceptual level [ISO87, Hal01]. Informally, we may define the conceptual level as being a level of system description which abstracts away from all decisions pertaining to the identification and usage of technology. In the case of an information system, one would typically position the information model [ISO87, Hal01], or conceptual schema, at this level.

At the other extreme, we find the physical level. At this level, one will typically see all technological details, dealing with a system's implementation. In other words, at this level, one will typically be interested in seeing the final result after all technological decisions have been made.

Sometimes a third level is introduced, the logical level. This level tends to be concerned with technological issues at a broader level. For example, rather than showing how an information model is mapped to a set of SQL tables of a specific SQL-based database management system using all performance optimisations that are specific to that platform, this level would show how the information model is translated to a relational database management system in general (as opposed to a specific instance). One might say that:

- the conceptual level is technologically independent,
- the logical level is bound to technology types (e.g. relational database management system),
- the physical level is bound to specific instances of technology.

To each of these levels, different viewpoints may be associated.

The above discussion, however, still does not provide a satisfying definition of implementation abstraction. What, for example, *is* technology? Does the fact that we decide whether or not to have a computer execute a certain process, as opposed to a human, constitute a technological decision? According to [Mer03], technology is:

1a the practical application of knowledge especially in a particular area,

1b a capability given by the practical application of knowledge,

2 a manner of accomplishing a task especially using technical processes, methods, or knowledge,

3 the specialized aspects of a particular field of endeavor.

When using this definition, most system models we would like to regard as conceptual, and supposedly technology free, are not technology free. So how can we define implementation abstraction? It is a form of abstraction that has proven its use (as well as pitfalls) in practice.

Rather than trying to come up with a final definition of different levels of implementation abstraction, we take the approach that abstraction from technology is inherently subjective. When applying implementation abstractions, it seems as if, at the back of our minds, we have some reference model of technological abstractions. We somehow seem to harbour some stratified reference model of less/more specific forms of technology. For example, most people would agree that the following phenomenon have a decreasing level of technological specificity:

1. Database management system 1020
2. Oracle database management system
3. Oracle database management system
4. Relational database management system
5. Relational database management system
6. Database management system

Given a set of system elements we presume a viewer's (or a group of viewers') reference model with regards to technological implementation to be provided as a function:

$$\mathsf{TechLevel} : \mathcal{MEL} \to [1..N]$$

Using this function, we may define the set of system elements that of level i as:

$$\mathcal{MEL}_i \triangleq \left\{ e \in \mathcal{MEL} \;\middle|\; \mathsf{TechLevel}(e) = i \right\}$$

The implementation levels as stipulated by TechLevel and the levels of abstraction in AbstractionOf should honour each other. To this end, we require the following axioms to hold:

[SY10] (*closed under abstraction*) $\forall_{1 \leq i < N} \, \forall_{x \in \mathcal{MEL}_i} \, \exists_{y \in \mathcal{MEL}_{i+1}} \, [x \; \mathsf{AbstractionOf} \; y]$

[SY11] (*closed under concretisation*) $\forall_{1 \leq i < N} \, \forall_{y \in \mathcal{MEL}_{i+1}} \, \exists_{x \in \mathcal{MEL}_i} \, [x \; \mathsf{AbstractionOf} \; y]$

We will refer to a system S as being at level n, with respect to some reference model TechLevel if n is the minimum of the levels of the elements in S:

$$\mathsf{SysTechLevel}(S) \triangleq \mathtt{min} \left\{ \mathsf{TechLevel}(e) \;\middle|\; e \in S \right\}$$

3.3 Towards a generic viewpoint

Towards *a* generic viewpoint – Why

- Approach a system domain carefully

  The more specific a language is,
  the more "conceptual prejudice" gets built into it

  Hammer ⇒ all pointy objects are nails

- Study system designs in general terms

  Patterns, properties, heuristics, ...

Towards *a* generic viewpoint – Uses

- Requirements & design
- Studying the context of a system to be developed/changed

  System domain
  ⊆ Domain of change
  ⊆ Domain of interest
  ⊆ Environment

  What is the essence of the domain under consideration?

Towards *a* generic viewpoint – What

- No silver bullet
- We *do* need to be specific enough to say usefull things
- Intended as a 'common core' of more detailed viewpoints
- As little as possible "conceptual prejudice"
- Stay close to the way we, as humans, discuss systems & domains
- In other words, in terms of concepts that come natural to us

  ... the ones we can find in natural language used 'in' the domain

System descriptions grounded in natural language

Not an uncommon approach:

- NIAM/ORM (Halpin & Nijssen)
- UML use cases (Jacobson, Booch & Rumbaugh)
- Demo (Dietz & Van Reijswoudt)
- KISS (Kristen)
- OOSA (Embley et. al.)
- ...

These are "semantically too specific" for our current purposes; too much "conceptual prejudice"

Active elements

- We consider open-active systems
- So there is activity going on
- A specific class of should therefore be:

  System activity – a system entity that is conceived of as changing parts of the universe.
- When we discuss the essence of these activities in natural language, or even in a specification language, we will directly or indirectly use *verbs*:

  updates, checks, produces, changes, computers, refines, ...

Activity participants

- Each activity will have several participants

  Actors – a system element that is conceived of as having some involvement in a system activity. This involvement is a special kind of system relationship, referred to as an activity participation. Formally, these entities are represented as: $\mathcal{ACT} \triangleq \{ From(p) \mid p \in \mathcal{APN} \}$.
- A subset of the set of system relationships are therefore the activity participations:

  Activity participations – a system relationship between a system activity and one of its actors.

Activity participants

We can now define:

The set of activities: $\mathcal{SAC} \triangleq \{ To(p) \mid p \in \mathcal{APN} \}$

The set of participants: $\mathcal{ACT} \triangleq \{ From(p) \mid p \in \mathcal{APN} \}$

Classes of participation

- Some participations will be more active than others
- This is where we find inspiration in theories regarding verbs and the 'things' that may play a (functional) role in these verbs
- We limit ourselves to those classes that are indeed relevant when considering activity in systems

Classes of participation

In decreasing scale of activity

- Agentive participation – An activity participation, where the participant is regarded as carrying out the activity.

  For example: A *person* writing a letter to a loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.

  Two sub-classes may be identified:

  - Initiating participation – An agentive participation, where the participant is regarded as the initiator of the activity.
  - Reactive participation – An non-initiating agentive participation.

Classes of participation

- Instrumental participation – An activity participation, where the participant is regarded as being an instrument in the activity.

  For example: A person writing a letter to a loved one, at a *desk* in a romantically lit room, on a mid-summer's day, using a *pencil*, while the cat is watching.

Classes of participation

- Experiencing participation – An activity participation, where the participant is regarded as experiencing/undergoing the activity.

  For example: A person writing a *romantic letter* to a *loved one*, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the *cat* is watching.

  Three sub-classes may be identified:

  - Patientive participation – An experiencing participation, where the participant is regarded as purposely undergoing changes (including its very creation)
  - Receptive participation – An experiencing participation, where the participant is regarded as the beneficiary/receipient of the results of the activity
  - Observative participation – An experiencing participation, where the participant is regarded as observing/witnessing the activity

Classes of participation

- Locative participation – An activity participation, where the participant is regarded as being the location of the activity, in terms of a spatial or temporal orientation.

  For example: A person writing a letter to a loved one, *at the desk* in a romantically lit room, on a *mid-summer's day*, using a pencil, while the cat is watching.

- Catalysing participation – A circumstantial participation, where the presence of the participant is regarded as being beneficial (either in a positive or a negative way) to the activity.

  For example: A person writing a letter to a loved one, in a *romantically lit room*, on a mid-summer's day, using a pencil, while the cat is watching.

Subjective!!

- The distinction between the participation classes is highly subjective
- It depends ... on the system viewer
- People working *in* the domain are always right. Right?

Multiple roles & participations

- The classes of participation can be used to classify the roles played by the participants

  For example, a participant involved in a catalysing participation may be referred to as a catalyst.

- One model element may indeed participate in more than one activity, and thus be part of multiple classes.

  The train '1492' was damaged when truck 'SF-VG-80' colided with it.

  Erik went to Amsterdam by train '1492'.

Responsibilities of system entities

- During system analysis and design, it is usefull to study the resposibilities and collaborations between system entities
- It provides a deeper understanding of what goes on in a system domain
- UML CRC (Class Responsibility & Collaborations) cards
  → ERC (Element Responsibility & Collaborations) cards
- Basic idea:

  > Describe for an elements (when it's relevant) what its responsibilities are towards other elements and its potential collaborations with other elements

Responsibilities of system entities

Example:

- A person writing a letter to a loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.
- What are the elements?
- What elements collaborate?
- Which element is responsible for what?

Responsibilities of system entities

What do you do when you design/model/analyse a system ..

- What elements shall I use?
- What collaborations will-there-be/are-there?
- What are the responsibilities of the elements?

Coordination

- Some activities aim to coordinate other activities.

  For example:

  - Management activities
    - Explicit
    - Management of a project
  - Triggering
    - Implicit
    - Sales of goods **triggers** ordering of new stock

Predications

- Some activities are really predications

  For example:

  - *Erik* is a *Professor*
  - *Erik* has *blue eyes*
  - $3 + 2$ is 5
  - Typical participation classes:
    - Subject/agentive
    - Predicate/instrumental

Information systems

- The 'contents' of a (computerised) information systems are intended to be a reflection of phenomena in (parts of) another system:
  - ER/ORM/NIAM → representation of facts (Person 'Erik' works for Department 'IRIS')
  - OO → placeholder objects (Person 'Erik', Train '1230', . . .)

A generic view?

- **Warning**:
  - Natural language also harbours 'conceptual prejudice'
  - Most, if not all, Indo-European languages are rather state oriented:

    'Charles the bold' stands on 'the floor'
  - Some North-American native languages are activity oriented:

    'Dances with wolves' stands on 'what supports us when walking'
  - Imagine the impact on the way we view & design systems ....

    Is nature state oriented or a continuous flow of activities?

Modelling a system domain

- Activities → what goes on?
- Abstractions → from chaos to order!

Chapter 4

System Definition

Version:
28-03-2002

4.1 Understanding system needs

System development starts out from a situation in which there is some need with regards to a new system or a change to a pre-existing system. Before actually developing a system, it is sensible to first gain a better understanding of the specific system needs. These needs are are usually captured in terms of so-called requirements.

4.1.1 Requirements

The dictionary [Mer03] defines requirements to be:

- something wanted or needed
- something essential to the existence or occurrence of something else

The ISO IT vocabulary [ISO93] defines it to be:

- an essential condition that a system has to satisfy.

In defining requirements, the system to be designed is usually treated as a black box. However, we should realise that any system to be developed needs to fit within a pre-existing context. A context that may for example contain a pre-existing technological infrastructure, a component-system that should be used or a 'legacy' system. In other words, during the definition process, the system to be developed should indeed be treated as a black box, but as a box on top of other boxes. This is illustrated in figure 4.1.

In the case of a change to a pre-existing operational system, the requirements may also contain an identification of those parts of the pre-existing system that should remain unchanged.

4.1.2 System quality

In the past, in the context of software development, requirements focussed mainly on the question: *What should the system do?*. However, purely focussing on the desired behaviour of the system did not provide a deep enough understanding of the system needs to be really able to decide between design alternatives when it came to trade-offs between, for instance, flexibility and performance of a resulting system. This has lead to the introduction of the complementary question *How well should it do this?* that should be answered by

Figure 4.1: A system as a box on top of other boxes

the definition process as well. The field of "quality-oriented software engineering" tries to optimise, with regards to software systems, the use of qualitative requirements statements during the design.

Requirements can therefore be expressed more precisely if we define them in terms of desired qualities of a system:

Requirement – an essential quality property that a system or its system description has to satisfy.

The requirements may reflect qualities offered by the system to its environment, as well as qualities the system may/must use from its environment. Remember, in the definition process, we treat the system to be developed as a black box on top of other boxes.

Now we have defined requirements in terms of qualities, we need to more precisely define what we mean by qualities. In doing so we will base ourselves on [ISO01, ISO96]. The essence of quality of a system is captured in the following definition:

Quality – is the totality of systemic properties of a system that relate to its ability to satisfy stated and/or implied needs.

The qualities we referred to, are really an abbreviation of quality properties, which are a special class of systemic properties:

Quality property – is a special systemic property, used to describe and asses the quality of a system.

A system may *offer* quality properties to its environment to satisfy the needs of other systems. At the same time, a system will need to *use* quality properties offered by other systems in its environment for its own functioning.

In [ISO01, ISO96] a range of key classes of quality properties are identified. These classes are referred to as quality attributes:

Quality attribute – is a specific class of quality properties.

Each quality attribute may also have a number of sub-characteristics, zooming in on specific quality characteristics. In [ISO01, ISO96] the following quality attributes

Efficiency – is the relationship between the level of performance of the system (or a sub-system) and the amount of resources used, under stated conditions. Efficiency may be related to time behaviour (response time, processing time, throughput rates) and to resource behaviour (amount of resources used and duration of such use).

Functionality – bears on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

Sub-characteristics of functionality are: suitability, accuracy, interoperability, compliance, security and traceability.

Reliability – is the capability of the system (or a sub-system) to maintain its level of performance under stated conditions for a stated period of time. The mean time to failure metric is often used to assess reliability of systems. The reliability can be determined through defining the level of protection against failures and the necessary measures for recovery from failures.

Sub-characteristics of reliability are: maturity, fault tolerance, recoverability, availability and degradability.

Maintainability – bears on the effort needed to make specified modifications to the system (or a sub-system). Modification may include corrections, improvements or adaptation to changes in the environment, the requirements and the (higher levels of the) design.

Sub-characteristics of maintainability are: analysability, changeability, stability, testability, manageability, reusability.

Portability – is the ability of the system (or one of its components) to be transferred from one environment to another.

Sub-characteristics of portability are: adaptability, installability, conformance, replaceability.

Usability – bears on the effort needed for the use of the system (or one of its sub-systems) by the actors in the environment of the system.

Sub-characteristics of usability are: understandability, learnability, operability, explicitness, customisability, attractivity, clarity, helpfulness and user-friendliness.

4.1.3 A pragmatic context for requirements

Requirements on a system are put forward by different stakeholders [CGY99]. They can essentially be seen as a concretisation of the stakeholder concerns. This allows us to define:

Stakeholder requirements – a requirement posed on a future/changed system by a specific stakeholder.

These requirements should essentially be refinements of the stakeholder concerns.

Stakeholder requirements description – a (system) description of the stakeholder requirements on a future/changed system S from the perspective of a specific stakeholder, which covers:

- the requirements which will be imposed by the environment on system S and
- the requirements on other systems in the environment of system S which system S may/must use for its own purposes.

When the set of stakeholders is large enough, and in most practical situations this will be the case, the resulting sets of requirements are bound to contain inconsistencies. In other words, sets of requirements

that are put forward by different stakeholders that can be met in issolation, but when combined lead to a set of requirements that is practically unrealistic to meet. In this case, a trade-off needs to be made between the different requirements in search of a good compromise; a negotiation process.

Some of the stakeholders may be more important with regards to the new system than others. This is, however, a rather subjective issue. To make things more objective, it is helpfull to also know *why* a stakeholder has put forward a certain requirement. In other words: *Why should it do so?* The field of "goal-oriented software engineering" tries to optimise, with regards to software systems, the use of a deeper understanding of the motivations behind the requirements in the design of software systems that better meet the collective requirements of stakeholders.

The consistent set of requirements that result after negotiating/selecting between the stakeholder requirements is referref to as the set of system requirements:

System requirements – a requirement on a future/changed system. These requirements are usually an negotiated integration of stakeholder requirements and should essentially be refinements of the system mission.

System requirements description – a (system) description of the stakeholder requirements on a future/changed system S, which covers:

- the requirements which will be imposed by the environment on system S and
- the requirements on other systems in the environment of system S which system S may/must use for its own purposes.

The gap between these sets of requirements, is what needs to be 'designed in' by the design process.

Just as the set of stakeholder requirements is a concretisation of stakeholder concerns, the set of system requirements should provide a concretisation of the system mission.

Based on the discussion above, we can now refine figure 2.6 (page 36) to the situation as depicted in figure 4.2.

Figure 4.2: A system as a box on top of other boxes

4.1.4 Alignment

Alignment

Alignment – a system S_1 is aligned to a system S_2 if, and only if, system S_1 meets *all* requirements posed by S_2 on S_1.

This alignment may be strived for at different levels of refinement of a system design.

An interesting question is of course: *which viewpoints should be taken into account to achieve a good alignment?* Taking all viewpoints into account would indeed be a laborious task, if not sheer overkill. In practice, a selection needs to be made on the viewpoints that deserve the most attention in a specific situation.

What we argue is that the set of viewpoints that should be taken into account, as well as the set of relevant quality properties, are very much dictated by the situation at hand. For example, in knowledge-intensive application areas with informal and ad-hoc work processes, it may be wise to take the position of individual human actors (the human perspective) into account to achieve proper business-IT alignment. In other situations, however, for instance when dealing with well defined and explicit work processes, a work-process perspective may be of more relevance than a human perspective.

The alignment issue may now be translated to the question: *how can a sub-system be aligned to its super-system?* As a sub-system can be seen as a provider of services to the super-system, one reasonable way of looking at their alignment is to express this in terms of quality properties. The rationale is that the super-system will 'expect' the sub-system to meet certain quality criteria. The better the match between the quality as *expected* by the super-system and the quality as *offered* by the sub-system, the better the sub-system is aligned to the super-system. This also implies that whenever the alignment between a sub-system and a super-system needs to be evaluated, these quality requirements should be articulated.

A complicating factor is that over time, the quality criteria as demanded by the super-system are susceptible to change. In other words, even though x and y may be aligned at t, they may fall out of alignment when y evolves and consequently poses other quality properties on x. This requires x to evolve as well. To really allow the super-system and sub-system to co-evolve, the potential evolution of the quality properties should be taken into account as well. This may be done by not only taking the quality properties as posed by the super-system into consideration, but also the expected stability/evolution of these properties.

Two quality characteristics that maybe used to illustrate the need for business-IT alignment are adaptability (a sub-characteristic of portability) and changability (a sub-characteristic of maintainability). Due to the dynamic socio-economic environment in which organisations have to operate, they need the ability to quickly adapt themselves to this changing environment. This also requires that their (computerised) information systems are able to co-evolve with the organisational system. Typical examples of systems hampering changes to the surrounding organisational system are legacy systems. As a legacy system are, by their definition, hard to change, changes to the surrounding system are likely to be hampered by the legacy system.

4.2 Describing requirements

4.2.1 Requirement types

See also [MB01].

Requirements – types

Functional requirement – a requirement which is based on the functionality quality attribute.

Non-functional requirement – a requirement which is not a functional requirement.

Requirements – types

Product requirement – a requirement on any of the products, pertaining to a system as it will eventually result from a system development process.

Process requirement – a requirement pertaining to the actual system development *process*.

The latter class of requirements will be dealt with in part **??**.

Requirements – types

Run-time requirement – a product requirement on a system as it will eventually result from a system development process.

These requirements usually refer to quality attributes such as: functionality, usability, efficiency and reliability.

Note that these requirements are observable as systemic properties of the system resulting from the development process.

Development-time requirement – a product requirement on any of the products, pertaining to a system development process, other than the resulting system.

These requirements usually refer to quality attributes such as: maintainability and portability.

4.2.2 Appraisal of requirements

Requirements appraisal

- Negotiability, w.r.t. specific stakeholders
  - Low: Constraints
  - Medium
  - High: Nice to have
- Persistence over time, relative to estimated life-cycle of (component-)system

4.2.3 Good requirements

Objectives

- A stakeholder goal is an *objective*
- A requirement on a system is an *objective*

⇒ What is a good objective?

⇒ S.M.A.R.T. !!

S.M.A.R.T.

S pecific

M easurable

A ttainable

R ealisable

T ime bounded

S.M.A.R.T.

Specific Formulation should be: clear, consistent, simple and have the appropriate granularity

Measurable It is possible to measure whether the objective has been met

Attainable It is possible to achieve the objective at all

Realisable It is possible to achieve the objective within the situation at hand

Time bounded The objective must be achievable within a pre-defined time-span

See also [MK95].

S.M.A.R.T.T.

- In the case of refinements into -objectives, the sub*-objectives should be **T**racable to super-objectives
- For example: mission → requirements → -requirements

4.2.4 Requirements description languages

4.3 Eliciting requirements

4.3.1 Activities

Two key processes:

1. Elicit individual stakeholder requirements.
2. Negotiate an aggreed set of system requirements.

Finished in the above order, but step two may (will!) start before step one is finished.

4.3.2 Challenges

Challenges – Choose your viewpoint wisely

- Which viewpoint to use when first approaching a domain?
- The set of underlying concepts leads to a pre-conception with regards to the nature of the domain
- Already limits the design space!

Challenges – Requirements change

- Improved insight by informers as well as modellers
- Changes in the environment of the system to be developed
- Design decisions lead to new questions/insight wrt the requirements that need to be ascertained
- Elicitation and design are not likely to be linear phases in a strict sense

Questions

1. What the relationship between a requirement and a quality property?
2. What is the key difference between a run-time requirement and a development-time requirement?
3. Why are development-time requirements just as important as run-time requirements?
4. Why is it important to understand the goals of the stakeholders during the definition and design processes?
5. What difference may knowing the negotiability and/or the persistence of time of requirement make during system design?
6. Why is it sensible to make requirements SMART?

Recommended Reading

1. L. Chung, D. Gross, and E. Yu. Architectural design to meet stakeholder requirements. In P. Donohue, editor, *First Working IFIP Conference on Software Architecture (WICSA1)*, Software Architecture, pages 545–564, San Antonio, Texas, 1999. Kluwer.
2. R. Malan and D. Bredemeyer. Defining Non Functional Requirements. Technical report, Bredemeyer Consulting, 2001.
3. M. Mannion and B. Keepence. Smart requirements. *ACM Software Engineering Notes*, 20(2):42–47, April 1995.

Bibliography

[CGY99] L. Chung, D. Gross, and E. Yu. Architectural design to meet stakeholder requirements. In P. Donohue, editor, *First Working IFIP Conference on Software Architecture (WICSA1)*, Software Architecture, pages 545–564, San Antonio, Texas, 1999. Kluwer.

[ISO93] *Information technology – Vocabulary – Part 1: Fundamental terms*, 1993. ISO/IEC 2382-1:1993. `http://www.iso.org`

[ISO96] ISO. *Kwaliteit van softwareprodukten*. ten Hagen & Stam, Den Haag, The Netherlands, 1996. In Dutch. ISBN 9026724306

[ISO01] *Software engineering – Product quality – Part 1: Quality model*, 2001. ISO/IEC 9126-1:2001. `http://www.iso.org`

[MB01] R. Malan and D. Bredemeyer. Defining Non Functional Requirements. Technical report, Bredemeyer Consulting, 2001.

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003. `http://www.webster.com`

[MK95] M. Mannion and B. Keepence. Smart requirements. *ACM Software Engineering Notes*, 20(2):42–47, April 1995.

Chapter 5

System Design

Version:
26-03-2003

Way of modelling and communication for design.

5.1 Design description languages

System (design) description languages

- Techniques to denote designs
- Used as the way of modelling associated to a viewpoint
- System descriptions may be:
  - Informal or formal
  - Sketchy or specific
  - Incomplete or compelete
- You've seen several SDL's, in particular for (information) system requirements, in other lectures

5.2 System design as a refinement process

5.2.1 Refinement

From what to how

- A requirements view on a system to be developed will focus on *what* the system should do; it's desired systemic properties
- A high level design of the same system will focus on *how* this behaviour can be realised
- The high level design is also likely to define *what* needs to be done at a finer level of granularity
- A detailed design will identify *how* these latter things will be resolved

From what to how

- *What:* I want my business to be able to *always* serve our customers, during all of our opening hours
  How: Use system components that are 100% reliable in the systems processing requests from customers
- *What:* System components should be 100% reliable.
  How: Use proven technologies in system components
- ...

Refinements

- What *what* is, and what *how* is, is contextual
- One system view may be more general than another one
- If V_1 and V_2 are system views, then we may have:

  $$V_1 \,\mathsf{Refines}\, V_2$$

  V_2 is more *what* than V_1
- Designing a system, really involves a process of constant refinement of system views

5.2.2 Dimensions of refinement

Refinements in system views

- Time horizon – Strategical, tactical, operational
- Abstraction – Typing, Generalisation, Encapsulation, Implementation.
- Development life-cycle – Requirements, Design

5.2.3 Refinement levels

Putting it all together ...

Refinement levels

[ ]CL

Note that the actual development process, as it is suggested in the above diagram, is not likely to be linear along the solid lines.

5.2.4 Design decissions

An information system is developed to meet certain requirements. Ideally, it is provable that a given information system (design) meets its requirements. It is only obvious that there will be alternative designs that meet the (explicit!) requirements. Selecting among the alternative designs is where we find style in the artistic sense. It is where art should transcend the function (defined as the ability to meet the explicit requirements) of a system.

Design choices ...

- Suppose, during a design process, we have:

$$V_1 \,\mathsf{Refines}\, W \wedge V_2 \,\mathsf{Refines}\, W$$

- Based on what can we make a choice?
- Need more requirements?
  Enrich W in order to falsify one of the candidates?
- What if you cannot do so?
- That's one of the places where designing becomes an art!

5.3 Design principles

Design principles – What?

Design principles – a system description language, which is intended to aid in the transition from requirements to design.

Each principle is a predicate over the products of the design process, where a system is no longer treated as a black-box. It thus provides direction to the ensuing design process.

Design principles – Why?

[ ]CL

- Means of negotiation
- Early insight
  - into critical design decisions
  - missing requirements

Architecture principles

Architecture principles – a which is used to direct the design/architecting of an architecture.

Why architecture principles:

- Means of negotiation
- Early insight
  - into critical design decisions
  - missing requirements

Good design principles

Should:

- have a name
- have a clear scope (Aspect/Component system)
- give direction
- have a definition that is understandable by stakeholders

Good design principles

Should:

- have a motivation that is understandable by stakeholders
- be a tracable refinement of:
  - requirements
  - higher levels of system descriptions
- be S.M.A.R.T.

Example

Principle: All decisions concerning architecture should support the development of customer-oriented systems.

Scope: General

Motivation: Our company must increasingly present itself as a service provider. Communication with customers should be oriented on solving their problems.

Possible refinements:

- For each product, it should be known which problem it solves for the customer.
- Human-machine dialogues should be tuned to customer-oriented communication
- To recognize the customer's needs, an increasing amount of legal knowledge is required. This implies the use of knowledge technology.

Example

Principle: Tasks are accessible to automated users in all locations where users need to work to carry out their tasks effectively and efficiently.

Scope: Work processes

Motivation: Location independency increases efficiency and flexibility of work processes. If several work locations are available, this allows work to move in case of a (technical) problem.

Possible refinements:

- Functionality should not be bound to one particular workstation. It should be made available on groups of workstations, or through a central server or the mainframe.
- No single installations or stand-alone applications.

Example

Principle: Complexity in applications should be avoided through division of functionality in basic components and optional components.

Scope: Applications

Motivation: Small components are more transparent, can be built faster, and can be maintained better.

Possible refinements:

- Version management for components is a necessity.
- Components should be homogeneous in nature.

Granularity

Principles may have differing granularity

- A component should be homogeneous in nature

  So: no workflow functionality and data functionality in the same component

- Complexity in applications should be avoided through division of functionality in basic components and optional components
- All architecture-related decisions should support the development of customer-oriented systems

Principles & Negotation

- Principles come to life in conflict situations
  - We prefer proven technology, so we should choose mainframe-based solutions
  - We prefer flexible ICT solutions, so we should choose a middleware-based solution, using multiple servers.
- Where does your top priority lie?
  - Give direction
  - Negotiate
  - Be aware of the choices you make!

5.4 Responsibility based design

Responsibilities of system entities

- During system analysis and design, it is usefull to study the resposibilities and collaborations between system entities
- It provides a deeper understanding of what goes on in a system domain
- UML CRC (Class Responsibility & Collaborations) cards
  → ERC (Element Responsibility & Collaborations) cards
- Basic idea:

  Describe for an elements (when it's relevant) what its responsibilities are towards other elements and its potential collaborations with other elements

Responsibilities of system entities

Rules of thumb:

- Do stick to the higher levels of abstraction!
- Focus on relevant elements, responsibilites & collaborations
- What are the relevant ones?
- Well ... modelling *is* an art!

Responsibilities of system entities

Example:

- A person writing a letter to a loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.
- What are the elements?
- What elements collaborate?
- Which element is responsible for what?

Responsibilities of system entities

What do you do when you design a system ..

- What elements shall I use?
- What collaborations will there be?
- What are the responsibilities of the elements?

Questions

1. What are the potential uses of architecture/design principles during system design?
2. What effects may the formulation of principles have on the requirements formulated thus far?
3. Take a sample ORM/NIAM/ER schema, with say, three object types connected by two relationship/fact types, also involving a subtyping between two of the object types. Also provide a small sample population of about three instances per relationship/fact type.

   Represent this sample schema and sample population in terms of

   What abstraction mechanisms did you use?
4. Take a sample activity diagram, in a familiar diagramming technique, containing two main activities and about three sub-activities per main activity. Also provide a small process-trace of an execution of this activity.

Represent this sample activity diagram and process-trace in terms of
What abstraction mechanisms did you use?

5. Why do we use abstraction mechanisms for system descriptions?
6. Where do we see relation conformity in data modelling (e.g. using ORM/NIAM/ER).

Complete Bibliography

[Avi95] D.E. Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, USA, 2nd edition, 1995. ISBN 0077092333

[BCK98] L. Bass, P.C. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, Massachusetts, USA, 1998. ISBN 0-201-19930-0

[Bem98] T.M.A. Bemelmans. *Bestuurlijke Informatiesystemen en Automatisering*. Kluwer, Deventer, The Netherlands, 7th edition, 1998. In Dutch. ISBN 9026727984

[Boa99] B.H. Boar. *Practical steps for aligning information technology with business strategies*. Wiley, New York, New York, 1999. ISBN 0471076376

[BP88] B.W. Boehm and P.N. Papaccio. Understanding and controlling software costs. *IEEE Transactions of Software Engineering*, 14(10):1462–1477, October 1988.

[Bro95] F. Brooks. *The Mythical Man-Month; anniversary edition*. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201835959

[CGY99] L. Chung, D. Gross, and E. Yu. Architectural design to meet stakeholder requirements. In P. Donohue, editor, *First Working IFIP Conference on Software Architecture (WICSA1)*, Software Architecture, pages 545–564, San Antonio, Texas, 1999. Kluwer.

[Che81] P. Checkland. *Systems thinking, systems practice*. John Wiley & Sons, New York, New York, USA, 1981. ISBN 0-471-27911-0

[Coc01] S. Cochran. The rising cost of software complexity. *Dr. Dobb's Journal*, April 2001.

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, 1998. ISBN 3-901-88201-4

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1-55860-672-6

[Ham90] M. Hammer. Re-engineering work: don't automate, obliterate. *Harvard Business Review*, 68(4):104–112, April 1990.

[HM95] J.H. Holland and H. Mimnaugh, editors. *Hidden Order : How Adaptation Builds Complexity*. Perseus Press, Cambridge, Massachusetts, 1995. ISBN 0201442302

[HV93] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4–16, 1993.

[HW92] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, September 2000. ISBN 0-738-12518-0
`http://www.ieee.org`

[ISO87] *Information processing systems – Concepts and Terminology for the Conceptual Schema and the Information Base*, 1987. ISO/TR 9007:1987.
`http://www.iso.org`

[ISO93] *Information technology – Vocabulary – Part 1: Fundamental terms*, 1993. ISO/IEC 2382-1:1993.
`http://www.iso.org`

[ISO96] ISO. *Kwaliteit van softwareprodukten*. ten Hagen & Stam, Den Haag, The Netherlands, 1996. In Dutch. ISBN 9026724306

[ISO98] *Information technology – Open Distributed Processing – Reference model: Overview*, 1998. ISO/IEC 10746-1:1998(E).
`http://www.iso.org`

[ISO01] *Software engineering – Product quality – Part 1: Quality model*, 2001. ISO/IEC 9126-1:2001.
`http://www.iso.org`

[Kee91] P.W.G. Keen. *Shaping the Future - Business Design Through Information Technology*. Harvard Business School Press, Boston, Massachusetts, USA, 1991. ISBN 0875842372

[Ken84] F. Kensing. Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon-DeMarco Approach. In T.M.A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 325–338. North-Holland, Amsterdam, The Netherlands, 1984.

[LS80] B. Lientz and E. Swanson. *Software Maintenance Management – a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley, Reading, Massachusetts, 1980. ISBN 0201042053

[Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. (In Danish).

[MB01] R. Malan and D. Bredemeyer. Defining Non Functional Requirements. Technical report, Bredemeyer Consulting, 2001.

[McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003.
`http://www.webster.com`

[Min94] H. Mintzberg. *The Rise and Fall of Strategic Planning*. Free Press, New York, New York, 1994. ISBN 0029216052

[MK95] M. Mannion and B. Keepence. Smart requirements. *ACM Software Engineering Notes*, 20(2):42–47, April 1995.

[MR02] M.W. Maier and R. Rechtin. *The Art of System Architecting*. CRC Press, Boca Raton, Florida, 2nd edition, 2002. ISBN 0849304407

[NP90] J. Nosek and P. Palvia. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance*, 3(2):157–174, 1990.

[Ode00a] J. Odell. Agents (part 1): Technology and usage. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[Ode00b] J. Odell. Agents (part 2): Complex systems. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[Par71] D.L. Parnas. Information distribution aspects of design methodology. In C.V. Freiman, J.E. Griffith, and J.L. Rosenfeld, editors, *Information Processing 71, Proceedings of IFIP Congress 71*, volume 1 – Foundations and Systems, Ljubljana, Yugoslavia, August 1971. North-Holland. ISBN 0720420636

[PB89] M.M. Parker and R.J. Benson. Enterprisewide information management: State-of-the-art strategic planning. *Journal of Information Systems Management*, (Summer):14–23, 1989.

[PBHJ00] H.A. Proper, H. Bosma, S.J.B.A. Hoppenbrouwers, and R.D.T. Janssen. An Alignment Perspective on Architecture-driven Information Systems Engineering. In D.B.B. Rijsenbrij, editor, *Proceedings of the Second National Architecture Congres*, Amsterdam, The Netherlands, EU, November 2000.

[Pro94] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 90-9006849-X

[Pro98] H.A. Proper. Da Vinci – Architecture-Driven Business Solutions. Technical report, Origin, Amsterdam, The Netherlands, EU, Summer 1998.

[Pro01] H.A. Proper, editor. *ISP for Large-scale Migrations*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU, 2001. ISBN 9076304882

[PW94] H.A. Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.

[PW95a] H.A. Proper and Th.P. van der Weide. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.

[PW95b] H.A. Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[Sim62] H.A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, volume 106, pages 467–482, 1962.

[Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0-444-86705-8

[Sol88] H.G. Sol. Information Systems Development: A Problem Solving Approach. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.

[SWS89] P.S. Seligmann, G.M. Wijers, and H.G. Sol. Analyzing the structure of I.S. methodologies, an alternative approach. In R. Maes, editor, *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, The Netherlands, EU, 1989.

[SZ92] J.F. Sowa and J.A. Zachman. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3):590–616, 1992.

[Tap96] D. Tapscott. *Digital Economy - Promise and peril in the age of networked intelligence*. McGraw-Hill, New York, New York, USA, 1996. ISBN 0070633428

[TC93] D. Tapscott and A. Caston. *Paradigm Shift – The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993. ASIN 0-070-62857-2

[TM86] A.J. Thomson and A.V. Martinet. *A Practical English Grammar*. Oxford University Press, Oxford, United Kingdom, fourth edition, 1986. ISBN 3810905798

[Vel92] J. in 't Veld. *Analyse van organisatieproblemen – Een toepassing van denken in systemen en processen*. Stenfert Kroese, Leiden, The Netherlands, 1992. In Dutch. ISBN 9020722816

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multi-view Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0-632-01216-8

[WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, EU, 1990. Springer-Verlag, Berlin, Germany, EU. ISBN 3-540-52625-0

[WJK00] M. Wooldridge, N.R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[Zac87] J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

Glossary of terms

In the definition of terms, we may use yet other terms that are also defined in this glossary. To identify these terms we shall use the **boldface** font.

Active system – a special kind of **system** that is conceived of as begin able to change parts of the **universe**.

Activity participation – a **system relationship** between a **system activity** and one of its **actors**.

Formally, this is represented by a set $\mathcal{APN} \subseteq \mathcal{MRL}$.

Actor – a **system element** that is conceived of as having some involvement in a **system activity**. This involvement is a special kind of **system relationship**, referred to as an **activity participation**.

Formally, these entities are represented as: $\mathcal{ACT} \triangleq \{ From(p) \mid p \in \mathcal{APN} \}$.

Alignment – a **system** S_1 is aligned to a system S_2 if, and only if, system S_1 meets *all* **requirements** posed by S_2 on S_1.

This alignment may be strived for at different levels of refinement of a system design.

Architectural description – A **system description** documenting/describing an **architecture**.

Architecture principles – a which is used to direct the design/architecting of an **architecture**.

Architecture-driven information system development – architecture-driven system development of an **information system**.

Architecture-driven system development – system development, using **architecture** as a means to

- guide & control the design and evolution of the **system**,
- evaluate & compare different system alternatives,
- negotiate the **concerns** of the system's **stakeholders**,

with the aim of optimising the **alignment** of the resulting system with its relevant, technological, organisational and human context (systems).

Architecture – a **system model** of which the **system description**, the so-called **architectural description**, is used during **system development** to:

- express the fundamental **organisation** of the **system domain** in terms of **components**, their relationships to each other and to the **environment** and
- the principles guiding its evolution and design,

and which's explicit intend is to be used as a means:

- of communication & negotiation among stakeholders,
- to evaluate and compare design alternatives,
- to plan, manage, and execute further **system development**,

- to verify the compliance of a **system** implementation's.

Aspect-system – an aspect-system S' of a **system** S, is a **sub-system**, where the set of **model relationships** in S' is a proper subset of the set of relationships in S. This is captured formally by means of the relation $\sqsubset_a \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$, where:

$$S' \sqsubset_a S \triangleq S' \subseteq S \wedge (S' \cap \mathcal{MRL}) \subset S$$

Component-system – a component-system S' of a **system** S, is a **sub-system**, where the set of **model entities** in S' is a proper subset of the set of entities in S. This is captured formally by means of the relation $\sqsubset_c \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$, where:

$$S' \sqsubset_c S \triangleq S' \subseteq S \wedge (S' \cap \mathcal{MEN}) \subset S$$

Component – is an abbreviation of **component-system**.

Computerised information system – a **sub-system** of an **information system**, whereby all activities within that sub-system are performed by one or several computer(s).

Conception – that what results, in the mind of a **viewer**, when they interpret a **perception** of a **domain**.

Concern – is an abbreviation of **stakeholder concern**.

Construction process – a process aiming to realise and test a **system** that is regarded as a (possibly artificial) artifact that is not yet in operation.

Data – any representation in some language.

Data is therefore simply a collection of symbols that may, or may not, have some meaning to some **actor**.

Definition process – a process aiming to identify all requirements that should be met by the **system** and the **system description**.

In literature this process may also be referred to as requirements engineering.

Description process – the combination of the **definition process** and **design process**.

Description – the result of a **viewer** denoting a **conception**, using some language to express themselves.

Design principles – a system description language, which is intended to aid in the transition from requirements to design.

Each principle is a predicate over the products of the design process, where a system is no longer treated as a black-box. It thus provides direction to the ensuing design process.

Design process – a process aiming to design a **system** conform stated requirements. The resulting system design may range from high-level designs, such as an strategy or an **architecture**, to the detailed level of programming statements or specific worker tasks.

Development-time requirement – a **product requirement** on any of the products, pertaining to a **system development** process, other than the resulting system.

These requirements usually refer to **quality attributes** such as: **maintainability** and **portability**.

Domain – any 'part' or 'aspect' of the **universe**.

Dynamic system – a special kind of **system** that is conceived of as undergoing change in the cause of time.

Efficiency – is the relationship between the level of performance of the **system** (or a **sub-system**) and the amount of resources used, under stated conditions. Efficiency may be related to time behaviour (response time, processing time, throughput rates) and to resource behaviour (amount of resources used and duration of such use).

Element evolution – the evolution, over time, of a system element.

The set of system elements is captured formally as: $\mathcal{EEV} \triangleq \mathcal{TME} \rightarrowtail \mathcal{MEL}$.

Environment – the environment of a **domain** is that part of the **universe**, which influences that domain, or vice versa.

Note that an environment is indeed a domain itself.

Evolutionary approach – a (part of a) development process is executed completely in several iterations, leading to several consecutive versions of the set of deliverables.

Functional requirement – a **requirement** which is based on the **functionality quality attribute**.

Functionality – bears on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

Sub-characteristics of functionality are: suitability, accuracy, interoperability, compliance, security and traceability.

Goal – is an abbreviation of **stakeholder goal**.

Human actor – An **actor** which is a single human being, or essentially a set of human-beings, such as a team.

Incremental approach – a (part of a) development process is executed on a sub-system by sub-system basis, using some well-defined division of a system into sub-systems.

Information architecture – the **architecture** of an **information system**.

Note that an information system may range in scope from, e.g., a value-chain wide information system to a hand-held information-appliance.

Information system development – system development of an **information system**.

Information system – a **sub-system** of an **organisational system**, comprising the conception of how the communication and information-oriented aspects of an **organisation** are composed and how these operate, thus leading to a description of the (explicit and/or implicit) communication-oriented and information-providing actions and arrangements existing within the organisational system.

Information – the **knowledge** increment brought about when a **human actor** receives a message. In other words, it is the difference between the conceptions held by a human actor *after* interpreting a received message and the conceptions held beforehand.

Installation process – a process aiming to make a **system** operational, i.e. to implement the use of the system by its prospective users.

Interest – the specific reason(s) why a **system viewer** observes a **system domain**. This is usually a confluence of the **systemic properties** of interest to the **system viewer** and the aspects of the **system** that are considered relevant to these systemic properties.

Knowledge – a relatively stable, and usually mostly consistent, set of conceptions posessed by a single (possibly composed) **human actor**.

In more popular terms: "a human actor's picture of the world".

Linear approach – step by step execution of a (part of a) development process, where a consecutive step is not executed until the preceding step is finished.

Maintainability – bears on the effort needed to make specified modifications to the **system** (or a **subsystem**). Modification may include corrections, improvements or adaptation to changes in the environment, the requirements and the (higher levels of the) design.

Sub-characteristics of maintainability are: analysability, changeability, stability, testability, manageability, reusability.

Message – data that is transmitted from one **actor** (the sender) to another **actor** (the receiver).

A message may actually be 'routed' via several **actors** before reaching its actual receiver. For example, when **human actors** exchange messages, they usually need to make use of some other **actor** playing the role of a medium (for example, vibrations in the air, or an e-mail system).

Model description – the **description** of a **model**.

Model element – any part of a **model**.

The set of elements in a model is represented formally by the set $\mathcal{MEL}$.

Model entity – any **model element** that is not conceived of as a **model relationship**.

The set of model entities is formally represented by $\mathcal{MEN} \triangleq \mathcal{MEL} - \mathcal{MRL}$.

Model relationship – any **model element** that is conceived as a relationship between entities from the model.

Formally, the set of model relationships is represented as $\mathcal{MRL}$, where $\mathcal{MRL} \subseteq \mathcal{MEL}$.

Modelling technique – the combination of a **way of modelling** and a **way of communicating**.

Model – a purposely abstracted, clear, precise and unambiguous **conception** (of some **domain**).

Non-functional requirement – a **requirement** which is not a **functional requirement**.

Open system – a special kind of **dynamic system** that is conceived as reacting to external triggers, i.e. there may be changes inside the system due to external causes originating from the system's **environment**.

Organisational system – a special kind of **system**, being normally active and open, and comprising the conception of how an **organisation** is composed and how it operates (i.e. performing specific actions in pursuit of organisational goals, guided by organisational rules and informed by internal and external communication), where its **systemic properties** are that it responds to (certain kinds of) changes caused by the system **environment** and, itself, causes (certain kinds of) changes in the system environment.

Organisation – an administrative and functional structure.

Participant – is an abbreviation of **actor**.

Participation – is an abbreviation of **activity participation**.

Perception – that what results, in the mind of a **viewer**, when they observe a **domain** with their senses, and forms a specific pattern of visual, auditory or other sensations in their minds.

Portability – is the ability of the **system** (or one of its **components**) to be transferred from one environment to another.

Sub-characteristics of portability are: adaptability, installability, conformance, replaceability.

Process requirement – a **requirement** pertaining to the actual **system development** *process*.

Product requirement – a **requirement** on any of the products, pertaining to a **system** as it will eventually result from a **system development** process.

Qualities – is an abbreviation of **quality properties**.

Quality attribute – is a specific class of **quality properties**.

Quality property – is a special **systemic property**, used to describe and asses the **quality** of a **system**.

Quality – is the totality of **systemic properties** of a **system** that relate to its ability to satisfy stated and/or implied needs.

Reliability – is the capability of the **system** (or a **sub-system**) to maintain its level of performance under stated conditions for a stated period of time. The mean time to failure metric is often used to assess reliability of systems. The reliability can be determined through defining the level of protection against failures and the necessary measures for recovery from failures.

Sub-characteristics of reliability are: maturity, fault tolerance, recoverability, availability and degradability.

Requirements definition language – a which is used to define **requirements**.

Requirement – an essential **quality property** that a **system** or its **system description** has to satisfy.

Run-time requirement – a **product requirement** on a **system** as it will eventually result from a **system development** process.

These requirements usually refer to **quality attributes** such as: **functionality**, **usability**, **efficiency** and **reliability**.

Note that these requirements are observable as **systemic properties** of the system resulting from the development process.

Stakeholder concern – an **interest** of a **stakeholder**, resulting from the stakeholder's **goals**, and the role played by some **system**.

This usually pertains to the **system's development**, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

Stakeholder goal – the end toward which effort is directed by a **stakeholder**, in which the **system** (of which the stakeholder is indeed a stakeholder) plays a role.

This may pertain to strategic, tactical or operational end. The role of the system may range from passive to active. For example, a financial controller's goal with regards to a future/changed system may be to control **(system) development** costs, while the goal of users of the system may be to get their job done more efficiently.

Stakeholder requirements description – a **(system) description** of the **stakeholder requirements** on a future/changed **system** S from the perspective of a specific stakeholder, which covers

- the **requirements** which will be imposed by the **environment** on system S and
- the **requirements** on other systems in the environment of system S which system S may/must use for its own purposes.

  For example: the properties of a given infrastructure that must/may be used, 'legacy' systems with which S needs to interface, etc.

The gap between these sets of requirements, is what needs to be 'designed in' by the **design process**.

Stakeholder requirement – a **requirement** posed on a future/changed **system** by a specific **stakeholder**.

These requirements should essentially be refinements of the **stakeholder concerns**.

Stakeholder – a party (a **system viewer**) with a specific **interest** pertaining to a **system's development**, its operation or any other aspects that are critical or otherwise important.

Examples are: Users, operators, owners, architects, engineers, testers, project managers, business management, ...

Sub-system – a sub-system S' of a **system** S, is a system where the set of **model elements** in S' is a subset of the elements in S. Formally, since we identify systems by means of their underlying set of elements, this is defined as: $S \subseteq S'$.

System activity – a **system entity** that is conceived of as changing parts of the **universe**.

Formally, these entities are represented as: .

System architecture – the **architecture** of a **system**.

System description language definition – the definition of a **system description language**, comprising a combination of:

- A **way of thinking** identifying the perspective from which systems are viewed.
- A set of underlying modelling concepts as well as their inter-relationships.
- Syntax of the system description language.
- Semantics of the system description language.
- Notation technique(s) for the denotation of **system descriptions**.
- Analysis techniques for system descriptions that allow for verification/validation of several properties.
- Visualisation/Animation techniques for system descriptions.

System description language – a language that may be used to denote/analyse/visualise/animate **system descriptions**.

System description – the **description** of a **system**.

System development – a process involving the execution of four sub-processes: **definition**, **design**, **construction** and **installation** of a **system**. Processes that may be executed sequentially, interleaved, or in parallel.

System domain – a **domain** that is conceived to be a **system**, by some **viewer**, by the distinction from its **environment**, by its coherence, and because of its **systemic properties**.

System element – a **(model) element** of **system model**.

System entity – any **system element** that is not conceived as a **system relationship**.

System evolution – the evolution, over time, of a system.

The set of possible system evolutions is represented formally by the set $\mathcal{SEV} \triangleq \wp(\mathcal{EEV})$.

System mission – a role, to the benefit of the **goals** of **stakeholders**, for which the **system** is intended.

System model – a **model** of a **system**.

To distinguish between colloquial use of the term system and the more formal use of the fact that it is actually a model, we will use the term system model whenever we need to stress the fact that we refer to the fact that it actually *is* a model.

System relationship – any **system element** that is conceived as a relationship between entities from the model.

System requirements description – a **(system) description** of the **stakeholder requirements** on a future/changed **system** S, which covers:

- the **requirements** which will be imposed by the **environment** on system S and

- the **requirements** on other systems in the environment of system S which system S may/must use for its own purposes.

  For example: the properties of a given infrastructure that must/may be used, 'legacy' systems with which S needs to interface, etc.

The gap between these sets of requirements, is what needs to be 'designed in' by the **design process**.

System requirement – a **requirement** on a future/changed **system**. These requirements are usually an negotiated integration of **stakeholder requirements** and should essentially be refinements of the **system mission**.

System viewer – a **viewer** of a **system domain**.

System viewpoint – a specification of the conventions for constructing and using **system views**.

This involves: a **way of thinking**, a **way of modelling**, a **way of communicating**, a **way of working**, a **way of supporting** and a **way of using**

System view – a **model** of a **system domain** from the perspective of a related set of **interests** of a **system viewer**.

Systemic property – a meaningful relationship that exists between the **domain** of elements considered as a whole, the **system domain** and its **environment**.

System – a special **model** of a **system domain**, whereby all the things contained in that model are transitively coherent, i.e. all of them are directly or indirectly related to each other and form a coherent whole.

A system is conceived as having assigned to it, as a whole, a specific characterisation (a non-empty set of **systemic properties**) which, in general, cannot be attributed exclusively to any of its components.

See also **system model**.

Formally, a system is represented as subset of **model elements** ($\mathcal{MEL}$).

Universe – the 'world' under consideration.

It is presumed that the world consists of elements and relationships between these elements.

Usability – bears on the effort needed for the use of the **system** (or one of its **sub-systems**) by the actors in the **environment** of the system.

Sub-characteristics of usability are: understandability, learnability, operability, explicitness, customisability, attractivity, clarity, helpfulness and user-friendliness.

Viewer – a **human actor** perceiving and conceiving (part of) a **domain**.

See also **perception** and **conception**.

Viewpoint – is an abbreviation of **system viewpoint**.

Way of communicating – describes how the abstract concepts from the **way of modelling** are communicated to human beings, for example in terms of a textual or a graphical notation.

The way of communicating essentially forms the bridge between the way of modelling and the **way of working**, it matches the abstract concepts of the way of modelling to the pragmatic needs of the way of working.

Note that it may very well be the case that different are based on the same **way of modelling**, yet use different notations.

Way of controlling – the managerial aspects of **system development**. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management and governance (see [Ken84] and [Sol88]).

Way of modelling – identifies the *core concepts* of the **system description language** of a method, or a **system viewpoint**.

Way of supporting – the support to **system development** that is offered by (possibly automated) tools. In general, a way of supporting is supplied in the form of some computerised tool (see for instance [McC89]).

Way of thinking – articulates the assumptions on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* [Sol83, WAA85], *underlying perspective* [Mat81] or *philosophy* [Avi95].

Way of using – identification of heuristics that:

- define situations, classes of **stakeholders** and **interests**, for which a given **system viewpoint** is most suitable,
- provide guidance in tuning a given **system viewpoint** to specific situations, classes of **stakeholders** and their **interests** at hand. For example, in terms of the set of modelling concepts to be used, effective notations, visualisations, etc.

Way of working – structures (parts of) the way in which a system is developed. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.

Glossary of symbols

In the explanation of symbols, we may use yet other terms that are also defined in the glossary of terms. To identify these terms we shall use the boldface font. In the explanation of symbols, we may use yet other terms that are also defined in the glossary of terms. To identify these terms we shall use the **boldface** font.

$\mathsf{AbstractionOf} \subseteq \mathcal{MEL} \times \mathcal{MEL}$ – a relation expressing that a **system element** is an abstraction of another element. (Introduced on page .)

$\mathsf{AbstractionOf} \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$ – a relation expressing that a **system** is an abstraction of another system. (Introduced on page .)

$\mathsf{AbstractionOf}\_ \subseteq \mathcal{MEL} \times \wp(\mathcal{EEV}) \times \mathcal{MEL}$ – a relation expressing that a **model element** is an abstraction of another element, in the context of some specified set of element evolutions. (Introduced on page .)

$\mathcal{ABS} \subseteq \mathcal{MRL}$ – the set of abstraction relationships. (Introduced on page .)

$\mathcal{APN} \subseteq \mathcal{MRL}$ – a set of **system relationships**, representing the involvement of an **actor** in a **system activity**. (Introduced on page .)

$\mathcal{ACT}$ – a set of **actors**. (Introduced on page .)

$\sqsubset_a \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$ – a relation defining when one **system** is an **aspect-system** of another system. (Introduced on page .)

$\_@\_ : \mathcal{SEV} \times \wp(\mathcal{TME}) \rightarrow \wp(\mathcal{MEL})$ – a function yielding the **model elements** that are valid at some point of time in a **system evolution**. (Introduced on page .)

$\sqsubset_c \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$ – a relation defining when one **system** is an **component-system** of another system. (Introduced on page .)

$\mathcal{EEV}$ – the set of **element evolutions** (Introduced on page .)

$\mathcal{MEL}_i$ – the set of **system elements** at implementation abstraction level i. (Introduced on page .)

$From : \mathcal{MRL} \rightarrow \mathcal{MEN}$ – a function yielding the **model entity** that is the source of a **model relationship**. (Introduced on page .)

$Involved : \mathcal{MRL} \rightarrow \wp(\mathcal{MEN})$ – a function yielding the **model entities** that are involved in a **model relationship**. (Introduced on page .)

$\mathcal{MSG}$ – set of **messages** (Introduced on page .)

$\mathcal{MEL}$ – a set of **model elements**. (Introduced on page .)

$\mathcal{MEN} \subseteq \mathcal{MEL}$ – the set of **model entities**. (Introduced on page .)

$\mathcal{MRL} \subseteq \mathcal{MEL}$ – the set of **model relationships**. (Introduced on page .)

$\mathsf{RefinesTo} \subseteq \wp(\mathcal{MEL}) \times \wp(\mathcal{MEL})$ – a relation expressing the fact that one **system view** refines to another **system view**. (Introduced on page .)

$\mathsf{SysTechLevel} : \wp(\mathcal{MEL}) \rightarrow 1..N]$ – a function returning the level of implementation abstraction of a given set of **system elements**. (Introduced on page .)

$\mathcal{SAC}$ – a set of **system activities**. (Introduced on page .)

$\mathcal{SEV}$ – the set of **system evolutions**. (Introduced on page .)

$\mathsf{TechLevel} : \mathcal{MEL} \rightarrow [1..N]$ – a function returning the level of implementation abstraction of a given **system element**. (Introduced on page .)

$\mathcal{TME}$ – a set of points in time. (Introduced on page .)

$To : \mathcal{MRL} \rightarrow \mathcal{MEN}$ – a function yielding the **model entity** that is the destination of a **model relationship**. (Introduced on page .)

Subject Index

A

Active system, 29–31, 85

Activity participation, 56, 85, 88

Actor, 56, 85–88, 93

Alignment, 13, 16, 17, 47, 65, 85

Architectural description, 44–47, 85

Architecture, 11, 15, 17, 33, 44–47, 75, 85–87, 90

Architecture principles, 75, 85

Architecture-driven information system development, 17, 47, 85

Architecture-driven system development, 47, 85

Aspect-system, 30, 31, 39, 40, 86, 93

C

Component (see also: Component-system), 15, 25, 29, 44, 46, 63, 85, 86, 88

Component-system, 30, 31, 35, 42, 61, 86, 93

Computerised information system, 10, 11, 13–15, 31, 86

Conception, 26–29, 32, 34, 35, 45, 86, 88, 91

Concern (see also: Stakeholder concern), 17, 36, 37, 46, 47, 85, 86

Construction process, 11, 33, 86, 90

D

Data, 86, 88

Definition process, 11, 33, 34, 61, 62, 86, 90

Description, 27–29, 86, 88, 90

Description process, 11, 86

Design principles, 74, 86

Design process, 11, 33, 34, 64, 86, 89–91

Development-time requirement, 66, 86

Domain, 26, 27, 29, 31, 32, 48, 86–88, 90, 91

Dynamic system, 29, 86, 88

E

Efficiency, 63, 66, 87, 89

Element evolution, 33, 87, 93

Environment, 10, 15, 26, 27, 29, 31, 44, 46, 63, 64, 85, 87–91

Evolutionary approach, 34, 87

F

Functional requirement, 66, 87, 88

Functionality, 63, 66, 87, 89

G

Goal (see also: Stakeholder goal), 36, 87, 89, 90

H

Human actor, 26, 87, 88, 91

I

Incremental approach, 34, 87

Information, 9, 87

Information architecture, 47, 87

Information system, 9–12, 15, 17, 31, 39, 47, 53, 85–87

Information system development, 11, 33, 87

Information technology, 11–13, 16, 17

Installation process, 11, 33, 87, 90

Interest, 32, 35–37, 45, 48, 87, 89, 91, 92

K

Knowledge, 87

L

Linear approach, 34, 87

M

Maintainability, 63, 66, 86, 88

Message, 88, 93

Model, 27, 29, 30, 32, 40, 45, 88, 90, 91

Model description, 88

Model element, 30, 88, 90, 91, 93

Model entity, 30, 31, 86, 88, 93, 94

Model relationship, 30, 31, 86, 88, 93, 94

Modelling technique, 88

N

Non-functional requirement, 66, 88

O

Open system, 29, 31, 88

Organisation, 9–13, 15, 26, 31, 39, 44, 46, 85, 87, 88

Organisational system, 10, 11, 31, 87, 88

P

Participant (see also: Actor), 88

Participation (see also: Activity participation), 88

Perception, 26, 32, 86, 88, 91

Portability, 63, 66, 86, 88

Process requirement, 66, 88

Product requirement, 66, 86, 88, 89

Q

Qualities (see also: Quality properties), 62, 89

Quality, 62, 89

Quality attribute, 62, 63, 66, 86, 87, 89

Quality property, 62, 89

R

Reliability, 63, 66, 89

Requirement, 62–66, 85, 87–91

Requirements definition language, 89

Run-time requirement, 66, 89

S

Stakeholder, 17, 35–37, 40, 46–48, 63, 85, 89, 90, 92

Stakeholder concern, 36, 63, 64, 86, 89

Stakeholder goal, 36, 87, 89

Stakeholder requirement, 63, 64, 68, 89–91

Stakeholder requirements description, 63, 89

Sub-system, 10, 30, 31, 63, 86–91

System, 10, 11, 15, 25–27, 29–40, 42, 44, 46–48, 53, 61–66, 73, 85–91, 93

System activity, 56, 85, 90, 93, 94

System architecture, 33, 90

System description, 11, 18, 29, 33–35, 37, 39, 44–47, 53, 62–64, 85, 86, 89, 90

System description language, 90, 92

System description language definition, 90

System design, 35, 44

System development, 18, 35–37, 46, 47, 66, 85–92

System domain, 27–29, 31, 32, 34–36, 42, 45, 46, 85, 87, 90, 91

System element, 54, 56, 85, 90, 93, 94

System entity, 56, 90

System evolution, 33, 90, 93, 94

System mission, 36, 64, 90, 91

System model, 31, 32, 39, 46, 52, 54, 85, 90, 91

System relationship, 56, 85, 90, 93

System requirement, 35, 44, 64, 68, 91

System requirements description, 64, 90

System view, 26, 32, 37, 91, 94

System viewer, 26, 28, 29, 31, 32, 35, 36, 39, 87, 89, 91

System viewpoint, 37, 38, 40, 42, 48, 91, 92

Systemic property, 10, 27, 29, 31, 32, 62, 66, 87–91

U

Universe, 26, 29, 35, 56, 85–87, 90, 91

Usability, 63, 66, 89, 91

V

Viewer, 26, 27, 29, 31, 32, 48, 86, 88, 90, 91

Viewpoint (see also: System viewpoint), 37, 39, 42, 44, 45, 47, 91

W

Way of communicating, 17, 18, 37, 88, 91

Way of controlling, 18, 19, 35, 91

Way of modelling, 18, 19, 37, 88, 91, 92

Way of supporting, 18, 37, 91, 92

Way of thinking, 18, 19, 37, 90–92

Way of using, 37, 91, 92

Way of working, 18, 19, 37, 91, 92

Appendix A

Mathematical Notations

Version:
30-04-02

The mathematical notation used in this thesis is described briefly in this appendix.

A.1 Sets

As well as the set operations: $\cup, \cap, \setminus, \subseteq$ with their usual meaning, we also define:

$$\begin{array}{lcl} A \subset B & \triangleq & A \subseteq B \wedge A \neq B \\ A \not\subset B & \triangleq & \neg A \subset B \end{array}$$

The power set of a set A, i.e. the set of all subsets of A, is denoted as $\wp(A)$, where:

$$\wp(A) \triangleq \{B \mid B \subseteq A\}$$

Some sets will have a total ordering, for a finite set X with a total ordering $< \ \subseteq X \times X$ we can define the extremes:

$$\begin{array}{lcl} \mathtt{max}(X) & \triangleq & x \in X \textbf{ such that } \forall_{a \in X}\, [a < x \vee a = x] \\ \mathtt{min}(X) & \triangleq & x \in X \textbf{ such that } \forall_{a \in X}\, [x < a \vee a = x] \end{array}$$

A.2 Functions

A partial function f from A to B is defined by $f : A \rightarrowtail B$. Formally, it is a relation $f \subseteq A \times B$ such that $\langle a, b\rangle \in f \wedge \langle a, c\rangle \in f \Rightarrow b = c$. This property makes it possible to write $f(a) = b$ instead of $\langle a, b\rangle \in f$.

A total function f from A to B is defined by $f : A \rightarrow B$.

A.3 Relations

If $R \subseteq X \times X$ is a relation, then we will use: $x\, R\, y\, R\, z$ as an abbreviation of: $x\, R\, y \wedge y\, R\, z$.

Appendix B

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

B.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject

(or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

B.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

B.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

B.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

B.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

B.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

B.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

B.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

B.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

B.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.